AD704343

ARDC TR 5

ARDC

AD

TECHNICAL REPORT NO. 5

BRLESC I/II FORTRAN

by

Lloyd W. Campbell
Glenn A. Beck

March 1970

U.S. ARMY MATERIEL COMMAND
ABERDEEN RESEARCH AND DEVELOPMENT CENTER
ABERDEEN PROVING GROUND, MARYLAND

160

A B E R D E E N   R E S E A R C H   A N D   D E V E L O P M E N T   C E N T E R

TECHNICAL REPORT NO. 5

MARCH 1970

BRIESC I/II FORTRAN

Lloyd W. Campbell
Glenn A. Beck

Computer Support Division

Funded by all ARDC RDT&E Projects

A B E R D E E N   P R O V I N G   G R O U N D,   M A R Y L A N D

A B E R D E E N   R E S E A R C H   A N D   D E V E L O P M E N T   C E N T E R

TECHNICAL REPORT NO. 5

LWCampbell/GBeck/eff
Aberdeen Proving Ground, Md.
March 1970

BRLESC I/II FORTRAN

ABSTRACT

FORTRAN is a popular scientific programming language that has been implemented on many computers. This report describes the FORTRAN language in general and includes specific details about its implementation on the BRLESC I and BRLESC II computers at the Aberdeen Research and Development Center.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

I. INTRODUCTION

FORTRAN is a programming language that is widely used on a variety of computers and can be used on the Aberdeen Research and Development Center's (ARDC) BRLESC I and BRLESC II computers. FORTRAN was designed primarily for programming of scientific problems and the evaluation of arithmetic formulas.

This manual is intended primarily as a reference manual for programmers that are already familiar with FORTRAN and the BRLESC I/II computers; however, it includes a general description of the FORTRAN language and should prove helpful to anyone who is interested in writing or reading FORTRAN programs. Additional details and general information can be obtained from other FORTRAN manuals and publications.

This manual emphasizes the details, restrictions and special features of the language as implemented on BRLESC I and BRLESC II. FORTRAN is not exactly the same on all computers, and it is not exactly the same on BRLESC I and BRLESC II. Although the general rules are usually the same, differences in details do exist and some of these differences are quite subtle. It is relatively easy to write FORTRAN programs which when executed on different computers will yield different results. These differences may be due to differences in compilers or differences in the structures of the computers. However, most FORTRAN programs require relatively minor modifications to allow them to run on any given computer. The modifications usually require much less effort and time than would be required to reprogram the problem in another programming language.

There have been two prominent versions of the FORTRAN language. They are referred to as FORTRAN II and FORTRAN IV. FORTRAN IV does not include everything that was in FORTRAN II. However, the BRLESC I/II compilers have retained essentially all of FORTRAN II so that it will usually accept statements that are defined in either of these two versions of the FORTRAN language. A third "version" occurred in March 1966 when the American Standards Association (now called American Standards National Institute) published the document X3.9-1966 which describes

PRECEDING PAGE BLANK

the "standard" FORTRAN language. This standard language is essentially the same as FORTRAN IV and it was developed to promote interchangeability of FORTRAN programs between computers. Although the standard neither precise nor complete enough to insure interchangeability, it can be used as an aid in writing programs that will have a good chance of proper execution on a variety of computers. It is recommended that, whenever practical, programs be restricted to the standard statements and features. This not only makes it easier to run a program at another installation, but it will probably allow the program to run on either BRLESC I or BRLESC II and it will simplify the reprogramming task when ARDC obtains other computers. An attempt is made in this manual to indicate those statements and features that are nonstandard. In addition, any things labeled as applying specifically to BRLESC I/II are also nonstandard. (Most of these things are specific details that are not covered by the standard rather than being different from something that is in the standard.) BRLESC I is more nonstandard than BRLESC II because the BRLESC I compiler was written before the standard existed.

II. THE CHARACTER SET

FORTRAN allows the use of the twenty-six capital letters of the alphabet, the decimal digits 0 to 9 and the special symbols + - ( ) . * / , = $. (BRLESC II and some computers also allow ').

The card code for these characters is the same as normally used for BRLESC I/II and most other computers and is sometimes referred to as "BCD" code. (See Appendix E.)

Within hollerith constants and hollerith information in formats, BRLESC I/II allow the use of all 64 characters except BRLESC II does not allow # and \.

Within FORTRAN statements, the blank character is ignored except within hollerith constants and hollerith information in formats. Blanks are not required within any statements and the insertion of blanks does not change the meaning of the statement.

III. SYMBOLIC NAMES AND CONSTANTS

General Names

In FORTRAN, all symbolic names (other than statement numbers) must begin with a letter and, for variables, the first letter usually determines the type of number it represents. Names of variables that begin with I, J, K, L, M or N represent integer numbers unless they are declared to be of some other type in a type-statement. Names beginning with other letters (A-H, O-Z) represent real (floating point) numbers unless they are declared to be of some other type in a type-statement.

The length of symbolic names is restricted to six characters in the standard and on most computers. A few computers allow eight characters. If names longer than six characters are used on BRLESC I/II, the first five characters and the last non-F character will be used as the name.

Statement Numbers

Locations of statements (cols. 1-5 of FORTRAN statement cards) must be all decimal digits and thus look like integer numbers but are really symbolic locations of statements. Leading zeros and all blank columns are ignored. (Statement numbers must be less than 32768 for some computers, but not BRLESC I/II.) On BRLESC I/II, statement numbers may be written in place of a variable name by writing an S after the statement number, e.g., as an argument in a CALL statement, 33S would represent statement number 33.

Constants

1. Integer constants are written without a decimal point. An integer constant on BRLESC I/II may consist of 1 to 17 decimal digits. Some computers restrict integer constants to as few as four decimal digits. The values of integer variables on BRLESC I/II must be less than 2^{64} in absolute value except the divisor and quotient of integer divide operations must both be less than 2^{34} in absolute value.

2. Real (floating point) constants must be written with a decimal point or an exponent. They may consist of a decimal point with 1 to 17 decimal digits (on BRLESC I/II) and may be followed by an E or D and a signed decimal exponent. (The D indicates double precision constants, however BRLESC I/II uses the same precision of sixteen decimal digits for both single and double precision.) The decimal point is not required when there is an exponent. The BRLESC I/II range of floating point constants (and variables) is between 10^{155} and 10^{-155} approximately in absolute value with zero also allowed. Most computers have a more restricted range of numbers.

Examples: 1. , 4.21 , .2 , 51.6 E2 , .1E-3 , 3.1 D-1

3. Alphanumeric constants of ten or less characters are allowed on BRLESC I/II. They must be preceded by nH where n is the number of characters in the constant. Blanks are not ignored in the n columns after the H. For interchangeability, it is best to limit n to 4 or 6.

BRLESC II also allows the nonstandard form of enclosing alphanumeric constants in apostrophies with a maximum of ten characters between the two apostrophies, e.g., 'ABC'. The appearance of two consecutive apostrophies between the enclosing apostrophies is considered to be one character which is an apostrophe, i.e., to include one apostrophe within this type of alphanumeric constant, insert two apostrophies. For example, 'X''''' is really X'' and is actually three characters long.

4. The logical constants allowed are ".TRUE." and ".FALSE.". Note the use of a period at both the beginning and end of these constants.

5. BRLESC I/II do not provide for standard complex arithmetic and complex constants. On computers that allow complex constants, they consist of the form (r, i) where r is the real part and i is the imaginary part and both r and i are written as single precision real constants.

6. All octal constants are nonstandard. Octal constants must not contain more than 22 digits on BRLESC I/II and the value must be less than 2^{64}. Such constants are stored like integers at the right end of a word. Negative octal constants are not allowed. Octal constants are allowed only on FORTRAN II type boolean cards with a B in column one (which are not allowed on BRLESC II) and in DATA statements. In DATA statements only, the octal digits must be preceded by the letter O.

<u>Arrays</u>

Blocks of storage are referred to as arrays in FORTRAN and are declared in DIMENSION,COMMON or type-statements. Particular elements of arrays are identified by subscripts enclosed in parentheses following the name of the array, e.g., A(3) or B(I,J). One, two, or three dimensional arrays may be used. Subscription of variables is done by substitution (rather than addition) and the lower bound of all subscripts is one. Subscript arithmetic is allowed; BRLESC I/II allow any integer arithmetic expression that does not itself involve any subscripted variables. However, the most general expression allowed in standard FORTRAN is C * V $\pm$ C' where C and C' are integer constants and V is an integer variable. Specifically, the only forms allowed by the standard are; C' , V , V + C' , V - C' , C * V , C * V + C' , C * V - C'.

IV. ARITHMETIC EXPRESSIONS

The following symbols denote the following operations:

+	addition
-	subtraction
*	multiplication
/	division
**	exponentiation

The use of functions (routines with only one result) is also allowed by writing the name of the function in front of parentheses that enclose the arguments, e.g., SIN(X). (FORTRAN allows functions to have more than one argument and commas are used to separate the arguments.) The arguments may be arithmetic expressions.

The precedence of operations when not governed by the use of parentheses is

functions
**
* and /
+ and -

where the operations higher on the list will be performed before those that are lower on the list. Successive + and - operations or successive * and / operations will be performed from the left to the right. Parentheses may always be used to cause the operations to be done in any desired sequence. Successive exponentiations must always have parentheses to show the desired grouping.

The standard does not permit implied multiplication (although some versions do allow it and BRLESC I/II allow it after a right parenthesis).

All arithmetic within an expression must be one type (integer or real) except for integer subscripts, integer arguments and integer powers of exponentiation in real expressions. The standard does permit real (single precision) numbers to be combined arithmetically with double precision or complex numbers.

Parentheses must not be omitted at the ends of an expression. The number of left parentheses must be the same as the number of right parentheses in each expression.

Two operations must not appear adjacent to each other in formulas; e.g., / - or ** - . The - and + operations may be used as unary operations at the beginning of an expression or after a left parenthesis, e.g., -A or (-A).

Any operation on integers which does not yield an exact integer result is truncated except negative integer results of division on BRLESC I will give the greatest integer that does not exceed the algebraic exact result. Thus -42/10 will give -5 on BRLESC I but should give -4.

From FORTRAN II on the 7090/7094, boolean expressions are allowed on cards with a B in column one on BRLESC I. The symbols +, *, - denote the logical operations of <u>or</u> (inclusive), <u>and</u>, and <u>not</u> respectively. BRLESC I performs these operations on the rightmost 65 bits of a word and the leading 3 bits of the word will be zeros after a logical operation. Note that FORTRAN IV has provided a new way of writing these logical operations as explained in Section VI below.

FORTRAN II double precision arithmetic expressions are allowed on BRLESC I (a D in col. 1) but are done in BRLESC I single precision which is as accurate as 7090/7094 double precision.

Complex arithmetic expressions are <u>not</u> presently allowed in BRLESC I/II FORTRAN. An I in column one or a complex type-statement will cause an error print.

V. ARITHMETIC ASSIGNMENT STATEMENTS

The general form of FORTRAN arithmetic assignment statements is

v = ae

where v is a name of an arithmetic variable (it may be subscripted) and ae is an arithmetic expression. An example would be

```
X(J + 1) = A(J)**2 - V/(T + 3.)
```

The arithmetic expression is evaluated and the result is stored as the new value of the variable whose name is on the left of the = symbol.

No arithmetic may be performed on the left of the = symbol except for subscript arithmetic. The standard and most computers allow only one = symbol and hence only one variable will have its value changed by an arithmetic formula.

However, BRLESC I/II allow up to 24 variables to be changed by the result of one arithmetic expression by writing arithmetic statements of the form vn = ... = v2 = v1 = ae where the result is stored in v1, v2, ..., vn and in that order, e.g., A = I = X is the same as two statements I = X followed by A = I.

If the type of the variable on the left of the = symbol is different than the type of the expression on the right of the = symbol, the value of the expression is automatically converted to agree with the type of the variable before it is stored.

The arithmetic expression may be just a name of a variable or constant, e.g., X = A.

On BRLESC I/II, if the arithmetic expression is a hollerith constant or contains a hollerith constant, the type of the variable v will not cause a conversion to be done. For interchangeability, it is best to use integer variables to contain hollerith information because an attempt to convert it to real will almost always cause the information to be changed. The standard does not allow hollerith constants to appear in arithmetic expressions although most computers do allow this.

VI. LOGICAL EXPRESSIONS

FORTRAN permits the use of logical variables and expressions that assume either the value .TRUE. or the value .FALSE.. The following three logical operations are defined using a and b to represent logical variables or logical expressions:

.NOT.a	is .TRUE. when a is .FALSE. and is .FALSE. when a is .TRUE.
a.AND.b	is .TRUE. when both a and b are .TRUE. and is .FALSE. when either a or b or both are .FALSE.
a.OR.b	is .TRUE. when either a or b or both are .TRUE. and is .FALSE. only when both a and b are .FALSE.

Two adjacent logical operations may be used only when the second one is .NOT.. Thus .AND..NOT. is legal but .NOT..AND. is illegal. The use of .NOT..NOT. is illegal but .NOT.(.NOT. is legal.

A relational expression that consists of a comparison of two arithmetic variables or expressions may be used to form logical expressions. FORTRAN uses the following relational operators: (x and y represent arithmetic variables or arithmetic expressions.)

x.EQ.y	is .TRUE. only if $x = y$.
x.NE.y	is .TRUE. only if $x \neq y$.
x.GT.y	is .TRUE. only if $x > y$.
x.GE.y	is .TRUE. only if $x \geq y$.
x.LT.y	is .TRUE. only if $x < y$.
x.LE.y	is .TRUE. only if $x \leq y$.

Whenever the relational expression is not .TRUE., it is .FALSE.. The arithmetic quantities x and y must be of the same type in any one relation unless one is real and one is double precision, e.g., if I is integer in I.LT.J, then J must also be integer.

On BRLESC I/II, the operands for .EQ. and .NE. could be logical variables but this is not true for most other computers.

It is illegal to use one arithmetic quantity as the operand for more than one relation. Hence the mathematical expression $x < y < z$ must be written as X.LT.Y.AND.Y.LT.Z and not as X.LT.Y.LT.Z.

A logical expression is any legal combination of logical operations and relational expressions. Parentheses may be used to obtain any desired grouping of operations. In the absence of parentheses, the operations are performed in the following order:

```
Arithmetic operations:  Functions
                           **
                         * and /
                         + and -
Relations:  .LT..LE..EQ..NE..GT..GE.
Logical operations:      .NOT.
                         .AND.
                         .OR.
```

Note that all the relations have equal precedence which means that they will normally be evaluated from left to right. Note also that .NOT. has a higher precedence than .AND. and .OR. and hence will be performed before the other two logical operations.

<u>Logical Assignment Statements</u>:

Logical expressions may be used in logical IF statements (see Section VIII, item 6) and in logical assignment statements. Logical assignment statements have the general form

v = le

where v is the name of a <u>logical</u> variable and le is a logical expression. The value stored in v will be .TRUE. or .FALSE. as determined by the evaluation of the logical expression le.

Examples of logical assignment statements:

(I,J,X and Y represent arithmetic variables and A,B and C represent logical variables.)

```
A = .FALSE.
C = A.AND..NOT.B
B = .NOT.(A.OR.B)
A = I.LE.3
B = I.EQ.J.AND.(B.OR.X.LE.Y)
C = 3.1416.GT.X+Y.OR.I*J.GT.1000
```

<u>Logical Masking Statements</u>:

To improve compatibility with CDC FORTRAN, BRLESC I/II allow non-standard logical masking statements. The operations .NOT.,AND., and .OR. may be used with <u>arithmetic</u> operands to accomplish bit-by-bit logical operations using the last 65 bits of BRLESC I words and all 68 bits of BRLESC II words.

An example of a logical masking statement would be

```
T = X.AND..NOT.Y
```

where X and Y are arithmetic variables (real or integer) and T may be any type of variable. This example will do a bit-by-bit product of X and the complement of Y and will store this result in T without any conversion.

VII. SPECIFICATION STATEMENTS

This group of statements (DIMENSION, COMMON, EQUIVALENCE, EXTERNAL and type-statements) provides information to the compiler and may be used by the programmer to control the storage assignment of some or all of the variables. These statements do not cause any machine code to be generated for running the program; they only affect the way it is compiled.

1. DIMENSION Statement:

```
DIMENSION a(i), b(i1,i2), c(i3,i4,i5),...
```

where a,b,c are array names and the i's are integer constants or integer dummy arguments.

This statement is used to declare the names and maximum sizes of arrays. The maximum subscripts are enclosed in parentheses and they must be decimal integer constants except integer dummy arguments may be used in subprograms if the array being declared is also a dummy argument. (See SUBROUTINE statement description.) The minimum subscript is always taken to be <u>one</u>. One, two, or three dimensional arrays may be declared in any sequence.

Arrays may also be declared in COMMON and type-statements with only one declaration allowed for the same array. BRLESC I requires that the very first appearance of an array name must be its declaration.

Example: `DIMENSION T(41),X(10),E(4,4,4),A(3,7)`

2. COMMON Statement:

```
COMMON a,b,c,d,e,...
```

where a,b,c,d,e are the names of variables of any type. Dummy argument names are not allowed.

This statement allows the programmer to specify that certain variables and arrays are the same in more than one program or subprogram (subroutine or function). The storage a signed to those items in the COMMON statements in one subprogram is the same storage assigned to the items in the COMMON statements in each of the other subprograms and the main program. Thus it also has an equivalence effect between subprograms. Note that correspondence is by storage and not by name, i.e., variables of the same name in different subprograms are the same only if they are assigned the same storage. All storage used in each subprogram is different than the storage in any other subprogram except for the items that are listed in COMMON statements.

Within each subprogram, all COMMON variables are assigned consecutively in the sequence in which they appear. The starting point for all the subprograms within each total program is the same. Proper space is left for arrays.

COMMON statements are used to avoid listing many arguments when using a subprogram. By forcing the main program and subprograms to use the same storage for some (and possibly all) of the variables, the need for specifying and moving variables is removed.

If any COMMON variable also appears in an EQUIVALENCE statement, the COMMON assigning has priority and is done first.

Dimension information may be specified in COMMON statements. However any one array must not be dimensioned more than once in the same program or subprogram, i.e., if an array name in a COMMON statement contains dimension information, it must not also be dimensioned in a DIMENSION or type-statement.

Standard FORTRAN and FORTRAN IV allow labeled COMMON blocks. A group of names may be preceded by a slash, a label name and another slash to give a name (label) to a section of the COMMON storage area. By using labeled COMMON, it is not necessary to think of COMMON as one big block. Whenever the same label is used in different subprograms, the corresponding members of the two labeled blocks will be assigned the same storage positions regardless of the relative position of the label within the respective COMMON statements. The following example will illustrate the meaning of labeled COMMON. If the first of the following COMMON statements appears in one subprogram and the second COMMON statement appears in a different subprogram within the same complete program,

```
COMMON A,X/LA/B,I,W/AA/P,M,N
..........
COMMON A, Y/AA/P,M1,N1/LA/E,J,W//Z
```

then the names A,P and W refer to the same quantity in both of the subprograms. The names X,B,I,M and N within the first subprogram refer to the same quantities respectively as the names Y,E,J,M1 and N1 in the second subprogram. In the second subprogram, the blank COMMON consists of A,Y and Z because two consecutive slashes cause the following quantities to be added to the blank COMMON block. Blank COMMON blocks do not have to be the same length in each subprogram. However labeled COMMON blocks of the same label must be the same length whenever they are used in different subprograms within the same complete program except BRLESC I only requires that the longest one appear first. (Length is defined as the amount of memory space used.) Any common block, including blank common, can be lengthened by an EQUIVALENCE statement except BRLESC I does not allow lengthening of labeled common blocks. Label names may be any legal FORTRAN name except the names of subroutines and functions may not be used. It is permissible to use the same name for a label and a variable within the same subprogram.

A subprogram may have more than one COMMON statement. Additional COMMON statements simply extend the list of COMMON variables. The use of the same label again within the same subprogram simply extends the list of variables in that labeled block. Thus the two consecutive statements

```
COMMON A,B,C/T/F,G
COMMON E/T/R,S//V
```

are the same as the single statement

```
COMMON A,B,C,E,V/T/F,G,R,S
```

On BRLESC I/II, the statements COMMON(USE MAIN) or COMMON(USE PREVIOUS) may be used instead of repeating long COMMON statements in a subprogram when all of the COMMON variables are identical with the main program or the previous subprogram. BRLESC II will assign double precision variables two storage positions when they appear in COMMON statements. However BRLESC I will assign only one storage position which can cause incorrect correspondence if a double precision variable was supposed to correspond to two real, integer, or logical variables.

3. EQUIVALENCE Statement:

EQUIVALENCE (a,b,c,...),(d,e,f,...),...

where a,b,c,d,e,f are names of any type of variable or subscripted array name. Dummy argument names are not allowed.

This statement causes different names to be assigned to the same storage space. All the names within a set of parentheses are made equivalent by assigning them the same storage space.

Subscripts on array names may be either a single integer constant or the proper number of integer constant subscripts, i.e., the correct number of dimensions. BRLESC I/II allow array names without subscripts to imply the first element of the array. BRLESC I also allows a single subscript on nonarray names; it acts like an increment when the storage is assigned and a subscript of one is the same as no subscript.

Whenever arrays are partially or completely overlapped, space is always reserved for all of the arrays involved so that there is no unexpected overlapping of storage. However, EQUIVALENCE will not rearrange COMMON storage; so equivalencing a larger array with a member of COMMON may cause additional overlapping of storage space.

It is illegal to use EQUIVALENCE to try to cause any impossible arrangement of storage. It cannot be used to attempt to cause nonconsecutive spaces to be assigned to elements of an array, to extend backward the beginning of the COMMON storage area or to equivalence two variables that are both in COMMON. It is also illegal for names of dummy arguments to appear in an EQUIVALENCE statement.

It is permissible to use EQUIVALENCE to extend the end of blank common or any labeled common block except BRLESC I does not permit such lengthening of a labeled common block.

On BRLESC I, it is illegal to equivalence anything to itself, either directly or indirectly.

On BRLESC I, any EQUIVALENCE statement that contains names of arrays and variables that are also in COMMON statements must appear after the DIMENSION and COMMON statements.

Example: EQUIVALENCE (A,B),(F(2,1),C,H(1))

4. Type-Statements:

Type-statements may be used to declare that specified variable names represent variables of a specified type. If a name does not appear in a type-statement, then its first letter determines whether it represents an integer or a real (floating point) number. However, a type-statement near the beginning of a program may be used to override (or confirm) the automatic type assignment or to declare a variable to be of some other type. A function name may appear in a type-statement if it is not the name of the subprogram containing the type-statement.

The following are type-statements:

```
INTEGER a,b,c,...
REAL a,b,c,...
DOUBLE PRECISION a,b,c,...
LOGICAL a,b,c,...
COMPLEX a,b,c,...
```

where a,b,c,... represents a list of variable and function names. On BRLESC I/II, DOUBLE PRECISION is used the same as REAL since double precision on most other computers is the same as BRLESC I/II single precision and the COMPLEX statement is not allowed because complex arithmetic is not allowed.

Variable array names in type-statements may also contain dimension information. However the same variable must not also be dimensioned elsewhere, i.e., it must not also appear in a DIMENSION statement or be dimensioned in a COMMON statement.

The names of all logical variables must be declared in a LOGICAL statement as there is no other method of distinguishing them from other variables.

The type-statements must precede all of the executable statements and DATA statements within each subprogram or main program. Note that type-statements are nonexecutable; they cannot be used between executable statements to cause any execution data conversion.

BRLESC I/II allow any of these type-statements to be preceded with the word TYPE because CDC FORTRAN allows this. It is for this reason that the names TYPEI, TYPER, TYPED, TYPEL and TYPEC must not be used as names of variables at the beginning of any statement on BRLESC I and at the beginning of the first executable statement (within each subprogram) on BRLESC II. (CDC and BRLESC I/II do not use the word PRECISION when DOUBLE is preceded by TYPE.)

BRLESC II will ignore an * and a decimal integer after the initial word of a type-statement and after a name, e.g., REAL * 8 M, R * 4, K is the same as REAL M, R, K.

Examples of type-statements:

```
REAL MASS, N2,LA(5,6),X
INTEGER A,F,I(15)
LOGICAL LV,T,WAY,LOW(18),NOW
```

5. EXTERNAL Statement:

FORTRAN requires this statement to be used whenever names of subroutines and functions are used as arguments for other subroutines or functions. It serves the same purpose as the card with F in column one did in some FORTRAN II compilers. The general form of the statement is:

```
EXTERNAL a,b,c,...
```

where a,b,c,... represents a list of function and subroutine names.

For BRLESC I, any statement function names used as arguments must also appear in an EXTERNAL statement. However, standard FORTRAN does not allow statement function names to appear in EXTERNAL statements or to be used as actual arguments. For BRLESC I, if the name of a function appears in both a type-statement and an EXTERNAL statement, the type-statement must precede the EXTERNAL statement.

Example:

```
EXTERNAL SIN,COS,FUN
```

VIII. CONTROL STATEMENTS

This group of statements provides for controlling the sequence in which statements are executed. Unconditional transfer of control, which is sometimes called branching or jumping, is provided by several types of GOTO statements and conditional transfer of control is provided by several types of IF statements. A DO statement allows definition of a "loop" and a CALL statement causes transfer of control to a subroutine with a return to the next statement. There are two statements, STOP and PAUSE, that cause the program to stop running. In the absence of control statements, the executable statements are executed consecutively in the order of their physical appearance beginning with the first executable statement of the main program.

1. GOTO Statement:

```
GOTO s
```

where s is a statement number. This statement causes the statement numbered s to be executed next.

Example: GO TO 22

2. Computed GOTO Statement:

```
GOTO(s1,s2,s3,...), i
```

where s1,s2,s3,... are statement numbers and i is a nonsubscripted integer variable. The statement executed next depends on the value of the variable i. If i = 1, statement s1 is done next; if i = 2, statement s2 is done next; etc. It is illegal for the value of i to be zero, negative, or larger than the number of statement numbers specified.

On BRLESC I/II, i = 0 causes the computer to cycle on one jump instruction that jumps to itself and an i that is too large causes the computer to jump to some statement or portion of statement that physically follows the computed GOTO statement. BRLESC I/II allows a maximum of 100 statement numbers to appear in this statement.

Example: GOTO(4,19,462),K

3. ASSIGN Statement:

```
ASSIGN s TO i
```

where s is a statement number and i is a nonsubscripted integer variable. This statement is used only in conjunction with the "assigned GOTO" statement (see 4. below) and is to be executed prior to the execution of the assigned GOTO statement. After execution, the value of i is not an integer number. On BRLESC I/II, it is the address of the statement numbered s.

Example: `ASSIGN 64 TO M`

4. Assigned GOTO statement:

```
GOTO i, (s1,s2,s3,...)
```

where i is a nonsubscripted integer variable and s1,s2,s3,... are statement numbers. This statement causes statement s1,s2, or s3 etc. to be executed next depending on which statement number was assigned to i by the previous execution of an ASSIGN statement. BRLESC I/II do not check whether or not the assigned statement number appears in the list and do not even require that the list appear although the list should be included as documentation and it is required by the standard.

```
Example:  ASSIGN 44 TO N
          GOTO N, (16,29,44,192)
```

5. Arithmetic IF Statement:

```
IF(ae)s1,s2,s3
```

where ae is an arithmetic expression and s1,s2, and s3 are statement numbers. This statement causes statement s1,s2 or s3 to be executed next depending on whether the value of ae is negative, exactly zero, or positive respectively.

```
Examples:  IF(X)4,7,22
           IF(R*V-4.1*(U+V))16,244,16
```

6. Logical IF Statement:

```
IF(le)st
```

where le is any logical expression and st is any executable statement except a DO statement or another logical IF statement. The statement st is executed if the value of the logical expression is .TRUE. and control simply goes to the next sequential statement if the value is .FALSE..

Examples:

```
IF(X.LT.5..AND.L.GE.70)GOTO 49
IF(I+J.EQ.14.OR.PRT)WRITE(2,16)A,B,C
   (where PRT is a logical variable)
```

7. Two Branch IF Statement:

```
IF(e)s1,s2
```

where e is either a logical or arithmetic expression and s1 and s2 are statement numbers. This statement is not standard but is allowed on BRLESC I/II and CDC computers. Statement s1 is executed next if e is .TRUE. (or nonzero for arithmetic expressions) and statement s2 is executed next if e is .FALSE. (or zero for arithmetic expressions).

```
Examples:  IF(X)22,471
           IF(X.GT.O..AND.L)962,1075
```

8. DO Statement:

```
DO s i = i1,i2,i3
```

or

```
DO s i = i1,i2
```

where s is a statement number, i is a nonsubscripted integer variable and i1,i2,i3 are unsigned integer constants or nonsubscripted integer variables that must be greater than zero when the DO statement is executed. The statement numbered s is called the terminal statement and it must physically appear somewhere after the DO statement. The sequence of statements that appear physically following the DO statement down to and including the terminal statement is called its range.

A DO statement causes its range to be executed repeatedly with the integer variable i initially assuming the value of i1. The variable i is incremented by i3 after each execution of the terminal statement and the sequence is repeated if the new value of i does not exceed i2. If i1 > i2 initially, CDC FORTRAN will not execute the loop even once whereas BRLESC I/II and most other computers will always execute a DO loop at least once. Standard FORTRAN does not permit i1 > i2 initially.

The specification of i3 is optional. If i3 is not specified, its value is taken as one.

The terminal statement must be executable and not any form of GOTO, arithmetic IF, RETURN, STOP, PAUSE, or DO Statement, nor any logical IF statement containing any of these statements.

For standard FORTRAN, the DO parameters, i,i1,i2 and i3, must not be changed within the range (or extended range) of the associated DO statement. BRLESC I/II will allow these parameters to be changed.

If a DO loop is exited by some statement other than the normal completion of the loop, the variable i will have its most recent value available for use. However, this variable is not available when the loop completes execution in the normal manner because not all computers will have i set to the same value. BRLESC I/II will have i set to the value that it would have had if the loop would have been executed one more time, i.e., the first value of i that exceeds i2 or i1 + i3, whichever is larger. The variable i is available within the range (and extended range) for use as either a subscript or integer variable.

In standard FORTRAN, it is illegal to transfer control into the range of a DO loop from outside its range (except for a return from an extended range), i.e. the DO statement must always be executed before any statements within its range are executed.

A DO range may contain other DO statements. However, any DO statement that appears within the range of another DO statement must terminate on or before the terminal statement of the DO statement that appeared first. More than one DO statement may use the same terminal statement.

If a group of statements are logically but not physically within the range of a DO statement, that group of statements is called an extended range. BRLESC I/II do not have any special restrictions on the use of one or more extended ranges. However the standard allows extended ranges only from the innermost range of a "completely nested nest" of DO loops and does not allow an extended range to contain any DO statements that have extended ranges. A completely nested nest is a group of DO loops, which must include all DO statements that are in the range of the first DO statement of the group, where all of the DO statements appear before any of the ranges are terminated and the first DO statement of the group is not in the range of any other DO statement.

Examples: DO 42 K = 1,L
DO 3 JT = MIN, 55, NSTEP

9. CONTINUE Statement:

CONTINUE

This is a dummy statement that generates no object code except when it is the terminal statement of a DO loop. It must be used as the terminal statement of a DO loop whenever the last statement would have been an arithmetic IF or GOTO type of statement that transfers control. Whenever a CONTINUE statement is the terminal statement of a DO loop on BRLESC I/II, its statement number is the location of the machine instructions that increment the DO variable and test for its maximum value.

10. STOP Statement:

STOP or STOP w

where w is an octal constant of not more than five digits that is ignored.

This statement causes the execution of a program to be terminated and should be used only to indicate that the program has run to completion. This statement causes BRLESC I/II to empty the tape output buffers, rewind all tapes used by the program that have not been rewound, check for overflows and halt at N40. On BRLESC I/II, the program may also be terminated by reading a card or tape line that has the first ten characters of either "ENDbTAPEbb" or "bbbbbbPROB" where b represents a blank.

Examples: STOP

STOP 77

11. PAUSE Statement:

PAUSE or PAUSE w

where w is an octal constant of not more than five digits.

This statement causes the program to halt and display the octal constant. (BRLESC I displays it in the α address of the halt order. BRLESC II displays it in the A register and in the address of the halt order.) If the computer is restarted manually by pressing the proper button (initiate on BRLESC I/II), the program will continue with the next statement.

This statement should not be used without a very good reason for using it.

Examples: PAUSE

PAUSE 421

12. CALL Statement:

CALL a(b,c,d,.....)

or

CALL a

This statement causes the subroutine named "a" to be entered and executed with b,c,d,... as the arguments. (Arithmetic expressions are allowed as arguments.) The subroutine being called must be one whose code is included in the program as a SUBROUTINE subprogram or one that is automatically made available by the compiler.

The arguments used in a CALL statement must agree in type with the type of the dummy arguments that were used when the subroutine was defined. If there are no arguments, they may be omitted.

CALL EXIT or CALL DUMP statements on BRLESC I/II are the same as a STOP statement and CALL PDUMP is ignored.

Alphanumeric constants of ten or less characters are allowed as arguments on BRLESC I/II.

BRLESC I/II allow "blank arguments" to be used by omitting an argument name and writing just a comma except that the last argument must not be blank. Blank arguments are actually optional arguments and may only be used with subroutines (and functions) that specifically allow them.

```
Examples:  CALL SUB3(X,Y,R)
           CALL TOTAL
```

13. Test Sense Switch Statement:

```
IF (SENSE SWITCH i) s1,s2
```

where i is an integer constant ($1 \le i \le 6$) and s1 and s2 are statement numbers.

This nonstandard statement transfers control to statement s1 or s2 if sense switch i is down or up respectively. On BRLESC I, the manual read switches 15-20 are used as sense switches 1 to 6 respectively. On BRLESC II, the manual read switches labeled 33, 37, 41, 45, 49, and 53 are used as sense switches 1 to 6 respectively; L is "up", any other value is "down". However, these switches may be preset by a program control card to be either "down" or "up" regardless of their actual position. (See SETSSW in Section XVIII.)

```
Example:  IF (SENSE SWITCH 3)14,92
```

FORTRAN IV does not usually allow this statement. Instead a subroutine SSWTCH is predefined. The general form of its use is

```
CALL SSWTCH (i,j)
```

where i is the number of the sense switch to be tested and j is set to 1 if it is down and j is set to 2 if it is up.

14. Set Sense Light Statement:

```
SENSE LIGHT i
```

where i is an integer constant ($0 \le i \le 4$). If i is 0, then all sense lights are turned off. If $1 \le i \le 4$ (actually 6 on BRLESC I), sense light i only will be turned on. The rightmost four (actually six) bits of cell 062 on BRLESC I are used as sense lights. Initially on BRLESC I, all of them are off. This is a nonstandard statement and is not allowed on BRLESC II.

Example: `SENSE LIGHT 2`

FORTRAN IV does not usually allow this statement. Instead, a subroutine SLITE is predefined. The general form of its use is

```
CALL SLITE(i)
```

where i is the number of the sense light to be turned on. If i = 0, all sense lights are turned off. BRLESC II allows the SLITE subroutine.

15. Test Sense Light Statement:

```
IF(SENSE LIGHT i)s1,s2
```

where i is an integer constant ($1 \le i \le 4$) and s1 and s2 are statement numbers. If sense light i is on, it is turned off and statement s1 is executed next; otherwise statement s2 is executed next. BRLESC I allows i to be as large as 6. This is a nonstandard statement and is not allowed on BRLESC II.

Example: `IF (SENSE LIGHT 2)67,39`

FORTRAN IV does not usually allow this statement. Instead, a subroutine SLITET is predefined. The general form of its use is

```
CALL SLITET(i,j)
```

where i is the number of the sense light to be tested and turned off. If the light was on, j will be set to 1 and if the light was off, j will be set to 2. BRLESC II allows the SLITET subroutine with $0 \le i \le 6$.

16. Test Overflow Statement:

IF ACCUMULATOR OVERFLOW s1,s2

where s1 and s2 are statement numbers. This nonstandard statement checks for floating point exponent overflow on BRLESC I and executes statement s1 next if it has occurred. Otherwise statement s2 is executed next. (The very last operation may not be included in the check on BRLESC I and this test turns the indicators off if they were on before.) BRLESC II always executes statement s2 next when this statement is executed. BRLESC II halts as soon as overflow occurs, therefore this statement is never executed after overflow has occurred.

FORTRAN IV does not usually allow this statement or the IF QUOTIENT OVERFLOW statement. Instead a subroutine OVERFL is predefined. The general form of its use is

CALL OVERFL(j)

where j is set to 1 if the overflow condition was on and j is set to 2 if it was off. The overflow condition is also turned off if it was on.

17. Test Quotient Overflow Statement:

IF QUOTIENT OVERFLOW s1,s2

where s1 and s2 are statement numbers. On BRLESC I/II, this nonstandard statement does exactly the same as the IF ACCUMULATOR OVERFLOW statement explained above.

18. Test Division Statement:

IF DIVIDE CHECK s1,s2

where s1 and s2 are statement numbers.

On BRLESC I, this nonstandard statement checks for floating point division by zero (or unnormalized divisor) or fixed point division overflow. If either has occurred in the program, statement s1 is executed next; otherwise s2 is executed next. (The very last operation in the previous statement may not be included in this test on BRLESC I and this test turns the indicators off if they were on before.) BRLESC II allows this statement, but it halts whenever a divisor is zero.

FORTRAN IV does not usually allow this statement. Instead a subroutine DVCHK is predefined. The general form of its use is

CALL DVCHK(j)

where j is set to 1 if either the BRLESC I floating or fixed point divide overflow condition is on and j is set to 2 if both are off. Both conditions are turned off if they were on. BRLESC II sets j to 2 and continues execution.

IX. FORMAT STATEMENT

FORMAT (Special Specifications)

This statement is not executed but is used to specify the field lengths, spacing and the form of the data for either the reading of input data or the printing (or punching) of output data. It is always used in conjunction with one of the input/output statements and does nothing by itself.

Let n = number of times to repeat this field. (n is optional, it is used as 1 if not specified.)

w = the width of the field (the number of columns or characters).

d = the number of decimal places to the right of the decimal point. Note that n, w, and d must be unsigned integer constants greater than zero. Then the types of fields that may be specified are:

nIw	for integer numbers.
nEw.d	for real numbers with exponents.
nFw.d	for real numbers without exponents.
wX	for spacing or blank columns.
nAw	for alphanumeric fields.
wH	for alphanumeric (Hollerith) fields where the characters are read into or printed from the w

	characters following the H in the FORMAT statement itself. BRLESC II also allows the use of ' * and $ characters to mark the beginning and end of hollerith information.
nLw	for logical variables.
nDw.d	for double precision numbers with exponents.
nGw.d	for generalized real numbers.
nOw	for Octal numbers. (nonstandard)
nRw	for right adjusted alphanumeric fields. (nonstandard)
nZw	for sexadecimal numbers on BRLESC II. (nonstandard)

Consecutive field specifications are separated by commas, thus "FORMAT (I6,3E14.6,F10.7)" is an example of a FORMAT statement. Each complete FORMAT statement specifies the maximum length of the record (card or printer line) that will be read, printed or punched when that FORMAT statement is used.

Three levels of parentheses are allowed in standard FORTRAN and four levels are allowed on BRLESC I/II so that groups of specifications may be repeated within a FORMAT statement. A left parenthesis may be preceded by an integer n to indicate the number of times to repeat the specifications enclosed in parentheses. Thus FORMAT (E12.5,3 (I6,F9.3)) would be a format where the I6,F9.3 portion would be repeated three times.

If the input/output statement list contains more items than specified by the format being used, then a new card or line is begun and the format is repeated from the left parenthesis that is associated with the next to last right parenthesis. (If there is only one pair of parentheses, then the format is repeated from the beginning.) If this parenthesis is preceded by a repeat number, it will be used. If the format specifies more fields than required for an input/output list, the rest of the format after the next field specification that would have required a name from the list is ignored provided the end of the format is not reached first. Note that X and H field specifications do not require a name from the I/O list.

A slash "/" may be used in a FORMAT statement to indicate that a new card or line should be started. Thus FORMAT (I10/E15.6) used for punching cards would cause a ten column integer to be on one card and a fifteen column real number to be on the next card. As a general rule, N consecutive slashes will cause N-1 blank lines or cards (or skip N-1 cards for input) except N slashes at the beginning or end of a format causes (or ignores) N blank lines. A format consisting of only N slashes will cause or skip N+1 blank lines. An empty format will cause or skip one blank line. Note that when one or more slashes are used between field specifications, the normal separating comma is not required.

Scale factors may be used with F type specifications (and in a limited way with E type specifications). An integer, s, specifies the power of ten (scale factor) to multiply the internal number by to obtain the external number, i.e., input numbers get divided by 10^s (not on BRLESC I) and output numbers get multiplied by 10^s. The integer s is written in front of the nFw.d specifications and the letter P is used to separate s and n, e.g., -2P4F10.5 or -2PF15.5 specify a scale factor 10^{-2}. Note that a minus sign is permitted to precede s but an explicit + sign is not permitted. Once s has been specified, the scale factor remains in effect for the rest of that FORMAT statement (including repetitions) and will be used on subsequent E and F type fields. A oP specification may be used to reset it to zero. For input, a punched exponent causes the scale factor to have no effect. For E fields on BRLESC I, only a positive scale factor may be used and it does not change the value of the number; it only indicates that s digits should be printed in front of the decimal point. (It has no meaning for input E fields.) Thus the number 2 would normally print 0.20E 01 for s = 0, but for s = 1, it would print 2.00E 00 and for s = 2, BRLESC I would print 20.00E-01, but BRLESC II would print

20.0E-01 which is the standard form. Thus for $s > 1$, BRLESC I prints a total of $s+d$ digits, but BRLESC II prints $d+1$ digits provided that $d \geq s-1$. For a negative scale factor in E fields on BRLESC II, $-s$ leading zeros are printed as part of the d digits after the decimal point which is as specified in the standard.

All output fields are right adjusted and are preceded with enough blanks to fill the field. The sign, if any, immediately precedes the numeric value. An all blank numeric input field will be converted to zero.

I Fields

Input: Most compilers assume the integer to be punched at the right end of the field without a decimal point; however, BRLESC I/II will accept it any place within the field and it may have a decimal point. Any digits following a point are ignored on BRLESC I/II.

Output: The integer will be punched at the right end of the field with a floating sign. (All output has a floating sign which means that the sign is in the column preceding the leftmost digit that is printed.) Leading zeros are not printed on I or F fields. If the integer is zero, a single zero is printed at the right end of the field. If the integer is too large for the field, BRLESC I prints an all blank field and BRLESC II prints asterisks.

E Fields

Input: The number may or may not have an exponent. An E or a sign, but not a blank, may be used to indicate the starting of an exponent. The exponent may be less than four columns. If a decimal point is punched, it is used and overrides the d specification. If no decimal point is punched, then it is assumed to be after d digits (columns) left from the start of the exponent. Most compilers require that the number be punched at the right end of the field, but BRLESC I/II allow it anywhere within the field. Blank columns are used as zeros (except after the exponent on BRLESC I/II). BRLESC II applies the scale factor only if the number does not have an exponent.

Output: The real number will be printed with a four column exponent that includes an E, a sign, and two digits for the value of the exponent. For exponents larger than 99, BRLESC I will use five columns for the exponent, BRLESC II will eliminate the E and print an explicit sign and a three digit exponent. If $s \leq 1$, a decimal point is printed d digits from the right end of the coefficient and if $s = 0$, a zero is printed in front of the decimal point. If $s \geq 1$, then s digits of the coefficient are printed to the left of the point and BRLESC I prints d digits after the point but BRLESC II prints d-s+1 digits to the right of the point. If $s < 0$, BRLESC II prints -s leading zeros within the d fractional digits. Note that the scale factor does not change the value of the number printed. The sign immediately precedes the first digit printed. The entire number is printed at the right end of the field of w columns.

F Fields

Input: The same as E fields, see above.

Output: The real number will be printed without an exponent and the decimal point will be printed d digits from the right end of the field. The actual number printed is 10^s times the number that is in the computer. If the number is too large for the columns specified, BRLESC I/II will print the number with an exponent or as much of the right portion of such a number as is permitted by the field width except BRLESC II will print asterisks if the width is less than four columns.

<u>H Fields</u>

<u>Input</u>: The alphanumeric information is stored in the FORMAT statement itself immediately following the H. No transformation of characters is done; the sign option setting for numeric input on BRLESC I/II has no effect on H fields.

<u>Output</u>: The w alphanumeric characters that immediately follow the H are printed. Blanks are not ignored. All 64 characters are permitted except BRLESC II does not permit # and \. However, other computers will probably not print the same character as BRLESC I/II if the character is not in the FORTRAN character set. For tape output, if an H field occurs at the beginning of a line, the first character is used by the printer for vertical spacing control instead of actually getting printed. The first character is always printed when the PUNCH statement is used on BRLESC I/II.

<u>' Fields</u>

BRLESC II allows apostrophies to mark the beginning and end of hollerith information in FORMAT statements, e.g., 'ABC' is the same as 3HABC. An apostrophe can be included within the string of characters by using two successive apostrophies to represent each one apostrophe that is to be included, e.g., 'X''''' would print as X'' but note that this would read five characters from an input line. Apostrophies in an input line are used just as they appear and therefore they must appear in pairs. The number of characters taken from an input line is determined by the number of characters that appear between the beginning and ending apostrophies.

* and $ Fields

BRLESC II allows * and $ characters to mark the beginning and end of hollerith information in FORMAT statements in the same manner as the apostrophe. However the same character must be used at both the beginning and the end and that character cannot appear within the string of characters.

A Fields

Input: BRLESC I/II stores a maximum of ten six-bit characters per word using the rightmost 60 bits of a word. If $w \leq 10$, w alphanumeric characters are stored in the variable specified by the input list. If $w < 10$, the characters will be at the left of the 60 bits with blanks to fill out the word. If $w > 10$, then $w - 10$ columns will be ignored before storing the rightmost ten characters of the field. As with H fields, no transformation of characters is done. This can be used to read FORMAT specifications into an array during execution.

Output: This causes w alphanumeric characters to be printed from the contents of the variable specified by the output list. The rules listed above for A input are followed so that whatever is read will be printed exactly the same. When $w > 10$, $w - 10$ blank columns will be printed to the left of the ten characters that are printed.

X Fields

Input: This causes w columns to be skipped whether they are blank or not.

Output: Causes w blank columns to be printed.

L Fields

Input: If the first non-blank character is a T (or the digit 1 on BRLESC I/II), the logical value .TRUE. is stored; otherwise .FALSE. is stored.

Output: A T is printed in the rightmost column of the field if the value of the logical variable is .TRUE.; otherwise an F is printed in the rightmost column of the field.

D Fields

Input & Output: This is allowed for those computers that use double precision variables. On BRLESC I/II, it is used exactly the same as an E field.

G Fields

Input & Output: BRLESC I/II use G fields exactly the same as F fields.

O Fields

Input: This allows octal numbers to be read and stored at the right end of BRLESC I/II words in the same manner as integers. On BRLESC I/II, if $w > 22$, the leading columns will be used and will cause more than 64 bits to be stored if they are not blank and it is illegal to store more than 64 bits.

Output: This allows integers (octal or decimal) to be printed in octal form at the right end of the field with leading zeros suppressed. If $w > 22$, w-22 blank columns are printed to the left of the 22 octal digits.

R Fields

Input & Output: Only a few computers and BRLESC I/II allow R fields. They are exactly like A fields except when $w < 10$, the characters are stored into (or printed from) the right end of the computer word.

Z Fields (BRLESC II only)

Input: If w ≤ 17, store w sexadecimal characters in one BRLESC II word. If w > 17, store the rightmost 17 sexadecimal characters of the field and ignore the other characters.

Output: If w ≤ 17, print the rightmost w sexadecimal characters. If w > 17, print all seventeen sexadecimal characters from one BRLESC II word in the rightmost 17 columns of the field. Leading zeros are suppressed.

FORMAT statements may be placed anywhere within a program (or subprogram) except as the last statement within a DO loop. (A few computers do not allow them immediately after a DO statement. On BRLESC I/II, FORMAT statements are done as NOP instructions so it is best not to place them where they will be done often.) FORMAT statements are kept as alphanumeric information and decoded at run time, thus it is permissible to use A fields to read FORMAT statements (without the word FORMAT) as hollerith input data during execution. The variable names of such object time formats must be declared to be an array. The non-subscripted array name may be used instead of a statement number for the format identifier in a READ or WRITE statement.

If the list in an output statement is exhausted and the next item in a format is an H field, the H field is printed. (If the end of the format and list occur at the same time and an H field follows at the rescan point, it will not be printed.) The scanning of the format actually precedes the scanning of the list except at the very end of the format. Therefore slashes and X fields will also get used from a format when they appear immediately to the right of the field specification that corresponds to the last item on the list. The format scan only stops when it reaches a field descriptor that requires a list item and the end of the list has been reached, or when both the end of the format and the end of the list have been reached. Note that a format may contain nothing but one or more H fields.

Blank characters in a FORMAT statement are ignored except within H fields or similar hollerith information. The w count for an H field must include the blanks within the H field.

On BRLESC I/II, the comma separating field specifications may be omitted when it follows an H or X field specification or would precede or follow a parenthesis or slash. The standard requires commas after H and X field specifications and a right parenthesis except before and after the final right parenthesis of the FORMAT statement. Commas are not required wherever one or more slashes appear.

```
Examples:  FORMAT(3I5,(E15.8))
           FORMAT(2HX=,F10.4,4(1PE12.5))
           FORMAT(6F10.4/4I10//)
```

X. INPUT/OUTPUT LISTS

The names of the variables to be transmitted between the computer and the input/output devices are specified in a list in the proper type of input/output statement and the sequence of the names in the list determines the sequence of transmission. Simple variable names, subscripted array names where the subscript control is either specified in other statements or within the input/output list, and array names without subscripts are allowed. Array names without subscripts cause the entire array to be transmitted and the elements must (for input) or will (for output) be arranged in the same sequence that they are in the computer memory. Arrays are stored such that the subscripts vary from left to right, thus two dimensional arrays are stored by columns; i.e., A(1,1), A(2,1), A(3,1) etc. is the sequence of elements of the array A. For dummy argument arrays, the number of elements is determined by the dummy argument array declaration rather than by the array declaration of the actual argument although BRLESC I uses the actual argument declarator. Commas are used to separate the names in an input/output list.

Indexing information specified within the list is written after the names of variables to which it applies and the names and the indexing information are all enclosed in parentheses. For example A (B(I),I = 1,10) would cause the transmission of A, B(1), B(2), ..., B(10). Note that the indexing information is written the same as in a DO statement with the increment taken as one if it is not written. It is permissible to nest these parentheses, e.g., ((A(I,J),I = 1,5), J = 1,5). Note that commas are used to separate items in the list and must be used after a right parenthesis except for the last one. The indexing within each set of parentheses is done to completion before going on to the next indexing specification. On BRLESC I/II, there is a restriction that when indexes are controlled within an I/O list, they cannot be used in any subscript arithmetic expression in that list that requires more than the addition or subtraction of a constant.

All of the input/output statements that transfer alphanumeric (not binary) data make use of format specifications to specify the field types and lengths. The type, e.g., integer or real, of a name specified in an input/output list must correspond to the type of field specified in the format that is being used. For example, all integer variables must use I fields. (BRLESC I/II do allow integers to be printed as integers in E or F fields.) The format controls the maximum length of each line. A line is shorter than specified in a format, only when the end of the list is reached before the end of the format. Whenever the end of the format is reached before the end of the list, the format is repeated from the left parenthesis that is associated with the next to last right parenthesis and a new line (or card) is started. (If there is only one pair of parentheses, then the format is repeated from the beginning.) (See Section IX for more information about FORMAT statements.)

Constants and arithmetic expressions are not permitted in I/O lists, except indexing information may contain constants and subscripts may be constant or arithmetic expressions. BRLESC I/II allow hollerith and positive decimal constants in I/O lists, but this is nonstandard.

It is permissible to read an integer variable and use it as a subscript within the same input list. However, BRLESC I requires that the integer variable name be separated from the place it is used by at least two left parentheses. (This is counting the one used to indicate a subscripted variable. Extra parentheses may be used just to meet this requirement.) Thus J,(B(J)) is an example where the value of the variable J just read will be used as the subscript for B(J). (For BRLESC I, the extra parentheses are not required if two or more variables or any indexing information separates the integer from where it is used.) BRLESC II does not require any extra parentheses and the above example could be written J,B(J).

```
Examples:  A, B, I
           N, M, (BA(N)),P
           ((A(I,J), J = 1,10), I = 1,10), (R(K), K = 2,20,2)
```

XI. INPUT/OUTPUT STATEMENTS

The following group of statements may be used to control the flow of information between the computer and input/output devices or secondary storage. Card reading or punching, magnetic tapes and, on some computers but not on BRLESC I/II, discs may be used to read or write data. Most of the statements also use a FORMAT statement, or its equivalent stored in an array, to control the conversion of data between computer form and printer or card form. However, the READ(t) and WRITE(t) statements cause the transfer of data without any conversion. This computer form of data will be referred to as binary information and actually is binary numbers for binary computers such as BRLESC I/II. The other statements cause the reading or printing of data in alphanumerical form. There are three statements, END FILE, REWIND and BACKSPACE that do not transfer data but can be used to manipulate the magnetic tapes.

In all of the input/output statements described below:

f	is a FORMAT statement number or array name.
list	is any allowable input/output list (See Section X).
t	is a magnetic tape integer constant or integer variable. (See BRLESC I/II restrictions on t at end of this section.)

1. Alphanumeric Read Statements:

```
READ(t,f) list
READ INPUT TAPE t, f, list (nonstandard form)
```

These statements cause decimal and alphanumeric input data to be read from tape t, converted according to the format specification identified by f, and stored in the variables specified by the list. Each block of BRLESC I/II tape may be as long as 2000 characters and each line may be as long as 160 characters. If the tape was previous FORTRAN output that has a vertical space control character at the beginning of each line, provision should be made in the format for skipping that character. However on BRLESC I/II, the vertical space control character is ignored unless the format has an H field at the beginning of the line. (If the tape was previous FORAST output, the vertical space control character is automatically ignored.)

The tape reading is parity checked and there is checking for end of reel.

If the "list" is omitted with this statement, it will cause at least one line to be read and ignored. More than one line will be ignored only if the format scan encounters a slash before it encounters a specification that requires a list item.

Just INPUT may be used instead of READ INPUT TAPE on BRLESC I/II.

2. Alphanumeric Write Statements:

WRITE (t,f) list

WRITE OUTPUT TAPE t,f, list (nonstandard form)

These statements cause decimal and alphanumeric output data to be written on tape t, after the variables specified by the list have been converted according to the format specification identified by f. Each line of data may not exceed a total of 132 characters. When a line of output is printed, the first character is used to control the vertical spacing of the paper and is not printed. BRLESC I/II insert an extra blank to indicate single spacing whenever the first character does not come from an H field (or ' * or $ field on BRLESC II). When the first character does come from an H field on BRLESC I/II, that character is used as a vertical space control character.

The tape writing is parity checked on BRLESC I/II and there is checking for the end of a reel. The number of lines per reel on BRLESC I/II will vary from about 70,000 to 200,000 as the length of each line varies from 132 characters to 1 character.

Just OUTPUT may be used instead of WRITE OUTPUT TAPE on BRLESC I/II.

3. Binary Read Statements:

READ(t) list

READ TAPE t, list (nonstandard form)

These statements cause binary information to be read from tape unit t and stored in the variables specified by the list. It should be used only for reading data that was previously put on tape by the use of the WRITE (t) statement described below. This statement will not read more data than was specified in the list of the statement that wrote the data. (Such a group of data is defined to be a "logical record".) If less than the entire logical record is read, the tape will move to the end of the record. (If the list is omitted entirely, the tape still moves to the next logical record.) If an attempt is made to read more data than is in one logical record, the unused portion of the list will be ignored on BRLESC I/II, but the standard does not allow the list to be longer than the record.

On BRLESC I/II, binary logical records are subdivided into tape blocks of 127 words each plus one extra word for a total of 128 words. Within each logical record, the first word of each block is zero except the first word of the last block contains the number of words in the last block (not counting the first word) and the total number of blocks in the logical record. The FORAST output of the BT.WR subroutine can be read by this FORTRAN statement.

4. Binary Write Statements:

```
WRITE(t) list
WRITE TAPE t, list (nonstandard form)
```

These statements cause all of the data specified in the list to be written as binary information in one logical record on tape unit t. It is useful for temporarily recording data on tape that may be read back into the computer by using a READ (t) statement at a later time. See the explanation of the READ (t) statement above for a description of the way the information is "blocked" on the tape on BRLESC I/II.

5. Write File Mark Statement:

```
END FILE t
```

This statement causes a file mark to be written on tape t. BRLESC I/II will ignore this statement when tape switch 8 has been specified and will give a run error print when tape switch 6 has been specified.

6. Move Backward Statement:

```
BACKSPACE t
```

This statement causes tape t to be moved backward one "logical record". This is all of the data written by the WRITE (t) statement that wrote the record for a binary tape, or is one line (or "card") if it is an alphanumeric tape. BRLESC I/II will give a run error print if tape switch 6 or 8 has been specified.

7. Rewind Statement:

REWIND t

This statement causes tape t to be rewound without being unloaded. It may be executed on BRLESC I/II when the tape is already rewound, but should be avoided as it requires time. BRLESC I/II will ignore this statement when tape switch 6 or 8 has been specified.

8. Read Cards Statement:

READ f, list

This nonstandard statement causes decimal and alphanumeric data to be read from cards (or tape switch 6 on BRLESC I/II if the cards have been put on tape off-line and the proper console switch is up.) If the list is omitted, at least one card will be read and ignored.

9. Punch Cards Statement:

PUNCH f, list

This nonstandard statement causes decimal and alphanumeric data to be punched on cards (or actual tape switch 8 on BRLESC I/II if the proper console switch is up). The BRLESC I/II tape output will be "formatted" for the high speed printer by adding a 1 character at the beginning of each "card" and an end-of-line character at the end of each "card". The block length will be at least 1830 characters. All 80 columns of a card may be used on BRLESC I/II and for tape switch 8 output, the "card" may be up to 132 columns long.

10. Print Statement:

PRINT f, list

For some computers, this nonstandard statement means to print the data on an on-line printer. Since BRLESC I does not have an on-line printer, the data is put on tape switch 8 for off-line printing. BRLESC II writes this data on tape switch 8 which, at the operators discretion, may also be printed on an on-line printer. The maximum line length for BRLESC I/II and for most computers is 132 characters.

The following description generally applies only for BRLESC I/II. If the first character of a line comes from an H field, it will be used for vertical space control (after a transformation) and not printed. If the first character does not come from an H field, an extra "1" character (single space) is inserted at the beginning of the line. The end-of-line character is automatically inserted at the end of each line. The tape writing is parity checked and there is checking for end of reel. The tape block length is at least 1830 characters and this allows about 15 million characters or 185,000 lines of 80 characters each on a reel of 1/2" tape.

For BRLESC I/II, a control card may be used to change all PRINT statements to PUNCH statements.

Additional Notes on Input/Output Statements:

The f (FORMAT number or name) may be omitted in READ, PUNCH or PRINT statements on BRLESC I/II and this will cause FORMAT (1P6E12.5) to be used automatically.

The statement numbers 1 and 2 may be used on BRLESC I/II to automatically specify FORMAT (5F14.5) and FORMAT (1P5E14.5) respectively without including them as part of the program. If 1 or 2 or both are used to refer to these formats, then that statement number must not be used in that subprogram for any other purpose. If either one is used as a statement number in a subprogram, then the corresponding automatic format cannot be used.

The omission of a "list" on any of the input statements will cause at least one record (card, line, or logical binary tape record) to be read and ignored on BRLESC I/II. More than one alphanumeric record may be skipped if the format contains slashes before the first specification that requires a list element.

The number of print positions on ARDC printers is 132.

ADDITIONAL NOTES ON THE USAGE OF MAGNETIC TAPE ON BRLESC I/II:

All of the tape reading and writing is parity checked. For one inch tapes, rereading erroneously ten consecutive times or rewriting wrong twice after each of five consecutive "GAP instructions" causes an error print and BRLESC I stops running the program. For half inch tapes, the unit halts the computer after sixteen unsuccessful rereads or rewrites.

There is checking for end-of-reel only on BRLESC I half inch tapes. At the end of a reel, BRLESC I halts at 080 and is ready to accept a new reel when restarted. A single reel of 1/2" tape will hold about 185,000 lines.

The only restriction on switching between reading and writing of tapes is that no reading can be done beyond the block that was last written on a tape, i.e., after writing, a tape must be moved backward before reading may occur. Whenever a tape on BRLESC I/II is switched from writing to reading, a file mark and an extra one word block that contains "END TAPE" is automatically written on the tape before the final file mark is written and then switching is done. (This extra block is ignored by a BACKSPACE statement.) BRLESC II can start writing after the use of the nonstandard subroutine BACKFILE. BRLESC I allows the use of BACKFILE, but will not properly start writing immediately after its execution. BRLESC I will also erase the file mark if writing begins after execution of the nonstandard subroutine SKIPFILE whereas BRLESC II will properly keep the file mark.

All FORTRAN alphanumeric input and output tapes are "buffered" and may contain up to 2000 characters per block. To accomplish this buffering, each tape unit used requires the use of an extra 201 words of BRLESC I/II memory. This space is assigned as it is needed while the program is being executed and will not conflict with any other memory assignment made in a normal FORTRAN program. Buffers are assigned backward from the subroutines provided space is available there. Otherwise they are assigned backward from the end of the memory. (For "CHAIN jobs" on BRLESC I, they are assigned so as to not conflict with any link of the CHAIN.)

FORTRAN programs are supposed to contain an END FILE statement and a REWIND statement for each output tape used in the program and a REWIND statement for each input tape. If this is not done within the program, BRLESC I/II will rewind all tapes that were used but not rewound by the program when the execution of the program has been completed.

Tape Unit Table.

The tape unit number t may be either a decimal integer constant or integer variable. If t is a variable, the integer value it has at the time the tape statement is executed is used as t. The following table shows the correspondence between the value of t and the tape switch number on BRLESC I/II. The actual physical tape handler used depends on the switch setting.

t	Switch
0	9
1 or 11	1
2 or 12	2
3 or 13	3
4 or 14	4
5	6 (Normal "card" input)
6	8 (Normal printer output)
7	10
8	11
9	12
10	7 (temporary or output only)
15	5

It is illegal to use both 1 and 11, or 2 and 12, or 3 and 13, or 4 and 14 within the same program. If t > 15 is used, it will be used modulo 16. PRINT (and PUNCH) tape output uses switch 8, the compiler itself uses switch 14 or 15 for its own program and may use switch 7 for temporary storage while compiling, and card input that was put on tape off-line uses switch 6. When leaving problems to be run on BRLESC I/II, the switch number rather than the t number must be used in the instructions to the computer operator.

Vertical Space Control.

All printer output is formatted for variable length lines for the off-line high speed printer. PUNCH tape output automatically has a single space character inserted at the beginning of each line and an end-of-line character at the end of each line. The same is true of PRINT and WRITE (t, f) output if the first field of the line is not an H type field. If the first field is an H field, then the first character of the field is used by the printer for vertical space control after undergoing the following transformation:

H Field	Tape
blank	1 (single space)
1	8 start new page
0 (zero)	special blank line with 1 on next line. (double space)
2	2 skip to even numbered line. (nonstandard) (possible double space)
8	8 (start new page) (nonstandard)
others	1 (single space) (nonstandard)

The vertical space control character is not printed and it causes the spacing to occur before the line is printed. A blank control character should normally be used to obtain single spaced lines. Some computers allow a + for no spacing, but this will cause single spacing on BRLESC

I/II. When reading previous FORTRAN alphanumeric output, a 1 vertical space control character is transformed back to blank and the special blank line is ignored but causes the 1 control character on the following line to be transformed back to zero. The 8 vertical space control character is not transformed back to 1, but note that the 8 will still start a new page if it is used for output. The special blank line is also ignored by the BACKSPACE statement.

XII. DECODE AND ENCODE STATEMENTS

ENCODE and DECODE are nonstandard statements that are allowed on BRLESC I/II and a few other computers. These statements on BRLESC I/II are generally compatible with CDC FORTRAN.

DECODE and ENCODE statements provide a means for performing I/O conversions under format control without actually using an I/O unit. The DECODE corresponds to normal input conversion except that it assumes the alphanumeric characters are already in the memory, either by a previous READ statement or by some other means. The ENCODE statement corresponds to normal output conversion except it stores the resulting alphanumeric characters in the memory instead of writing them on some output unit.

1. DECODE Statement:

```
DECODE (icr, f, ac) list
```

where icr is the integer number of characters per record, f is a FORMAT statement number or an array name, ac is a variable name, array name, or array element name that specifies the initial position of the alphanumeric characters that are to be converted according to the format f and the converted results are stored in the storage specified by the list. The list may be any normal I/O list.

BRLESC I/II assume that each memory word starting at ac, contains ten characters. However, the number of characters used from each word may be changed by execution of the SETCWD(i) subroutine where i is an integer less than or equal to ten that specifies the number of characters per memory word that will be used when subsequent DECODE statements are executed.

If the format and list require more than one record (line), the next record begins at the next memory word after icr characters, even if not all of the icr characters have been used. For example, if icr is 23 and there are 10 characters per memory position, then the second record would begin using characters from the beginning of the fourth memory word.

BRLESC I/II allow up to 160 characters to be decoded for each record, even when icr is smaller. It uses blanks after the last memory word that contains the icr characters. However, CDC does not allow more than icr characters to be decoded per record.

Examples:
```
DECODE (80,33,A)(B(I),I = 1,10)
DECODE (10,AF,E(3))X, J
```

2. ENCODE Statement:

```
ENCODE (icr, f, ac) list
```

where icr is the integer number of characters per record, f is a FORMAT statement number or an array name, ac is a variable name, array name, or array element name that specifies the initial position for storing the alphanumeric characters that result from converting the items on the list according to the format f. The list may be any normal I/O list.

BRLESC I/II will store ten characters at each memory word starting at ac and will always store at least icr characters. The number of characters stored per memory word may be changed by executing the SETCWE(i) subroutine where i is an integer less than or equal to ten that specifies the number of characters to be stored in each memory word when subsequent ENCODE statements are executed.

If the format and list specify less than icr characters in a record, the record is filled out with blanks and a total of icr characters are stored. If the format and list specify more than one record, the next record begins at the beginning of the next memory word after icr characters.

BRLESC I/II will encode up to 160 characters but only stores icr characters at ac plus any characters that will fit into the last memory word of the record. However, CDC does not allow encoding more than icr characters per record.

BRLESC I/II do not automatically insert any extra printer vertical space control character at the beginning of a record when ENCODE statements are executed.

```
Examples:  ENCODE (132,16,P(1))W, MT, K1, K2, F
           ENCODE (50,241,H) Q,((G(I,J),J = 1,2), I = 1,3)
```

XIII. DATA STATEMENT

The DATA statement allows initial values to be stored for variables without writing an executable formula. The DATA statement allows a list of variable names to be followed by a list of the constants that should be initially stored as the values of the variables. A slash is used to separate a list of variables and a list of constants and commas are used to separate items within both lists.

The general form of the DATA statement is

```
DATA v1,v2,v3,.../c1,c2,c3,.../,v4,v5,.../c4,c5,111/,...
```

where v1,v2,... represents names of variables and c1,c2,... represents constants. The variable list on BRLESC I/II may contain DO-implying parentheses with variable subscripts that take on specified integer constants, but the standard does not allow this. All other subscripts must be constant, i.e., the integer value of all subscripts must be completely defined within the DATA statement. The name of an array may be used without subscripts to specify a list of the entire array on

BRLESC I/II, but this is nonstandard. Dummy argument names are not allowed in DATA statements.

The constant list may contain any standard FORTRAN constant and may also contain octal constants on BRLESC I/II by preceding the octal digits with the letter O. T and TRUE are also allowed for .TRUE. and F and FALSE are also allowed for .FALSE. on BRLESC I/II. Any constant may be repeated k times by preceding it with "k*" where k is the integer number of times that the constant should be repeated.

Most computers and the standard do not allow the DATA statement to initialize a variable that is in blank COMMON; however this is allowed on BRLESC I/II. Also most other computers and the standard allow variables in labeled COMMON blocks to appear in DATA statements only within a special BLOCK DATA subprogram.

Some examples of DATA statements are:

```
DATA A,B/5.3,.6E-3/
DATA I,LOGIC,OCT/14.,FALSE.,O7777/,ALPH/4HDONE/
DATA(C(I),I=1,10)/5*1.0,3*2.0,2*3.0/
```

Note the absence of a comma after "DATA" but the presence of a comma before the beginning of any other list of variable names in the same statement.

There must be a one-to-one correspondence between the number of variables that are to be given initial values and the number of constants within any one DATA statement. BRLESC I/II gives an error print when there is not a one-to-one correspondence.

DATA statements may appear anywhere within a main program or subprogram except the standard and BRLESC II require that they must not appear before the last specification statement.

To allow some compatibility with CDC FORTRAN, BRLESC I/II also allows the CDC form of the DATA statement which has the general form of:

DATA(v1 = c1),(v2 = c2),...

where v1 is one variable name or one array name or one subscripted name, which may have DO-implying subscript information, and c1 is one constant or enough constants, separated by commas, to satisfy the requirements for v1. Repetition of one or more constants k times is allowed by "k(c1,c2,...)".

Some examples of CDC DATA statements are:

```
DATA(F=7.2),(X=.003)
DATA((B(J),J=1,5)=1.0,2(5.3,8.1)),(IA=.TRUE.)
```

XIV. SUBPROGRAM STATEMENTS

FORTRAN allows sections of a FORTRAN program to be designated as subprograms that may be used at many different places in the main program or in other subprograms. The SUBROUTINE, FUNCTION, RETURN, END, and BLOCK DATA statements allow the programmer to define and name portions of his program as subprograms and these statements provide information that allows the compiler to provide for the substitution of variables at execution time and to provide standard entry and exit methods. There are two kinds of executable subprograms, subroutines and functions. A BLOCK DATA subprogram is nonexecutable.

Any subprogram may use any of the FORTRAN statements within itself except SUBROUTINE, FUNCTION, and BLOCK DATA statements. Any executable subprogram may use any other subprogram or subroutine of any type, including statement functions (See Section XV) that are defined at the beginning of that subprogram. Recursive subprograms (subprograms that use themselves) are not allowed.

1. SUBROUTINE Statement:

```
SUBROUTINE a(b,c,d,e,...)
```

This statement marks the beginning of a subprogram that is called a subroutine. The name of the subroutine is a and b,c,d,e,...

are the names of nonsubscripted dummy arguments that will be replaced at execute time by the actual variables that are listed in the CALL statement that causes the subroutine to be executed. The subroutine consists of the FORTRAN statements that follow this statement down to an END statement. BRLESC I/II will also end a subprogram if it encounters a FUNCTION or another SUBROUTINE statement and do not require the END statement.

The name of the subroutine does not indicate the type of any result and hence any letter may be used as its first character.

Except for the common storage, all variables within a subprogram are assigned storage that is unique and not used by any other part of the program. Thus the variable X may be used in several subprograms within a program and each X will be different unless it appears in the same relative position in COMMON statements in each of the subprograms.

Example: SUBROUTINE FOGO(A,XX,LEM2)

Dummy Arguments and Adjustable Dimensions:

No storage is assigned to dummy arguments; on BRLESC I/II, DM will appear in the dictionary instead of a memory address. The type of a dummy argument, as indicated by its first letter or within a type-statement, must agree with the type of all actual arguments that replace it.

If a dummy argument is not an array, then an associated actual argument may be either a simple variable or a subscripted array name (which means that the subprogram will use just one element of an array as though it were a simple variable). A nonarray dummy argument may also be used within its subprogram as though it were a function name or a subroutine name provided that any associated actual argument is a function name or a subroutine name respectively.

If a dummy argument is an array, then any associated actual argument must also be an array and may be subscripted except it may not be subscripted on BRLESC I. The dummy and actual arguments must also have the same number of dimensions on BRLESC I, but not on BRLESC II or according to the standard. The dummy array need not be the same dimensions as the actual argument, but when the dimensions are not the same, the normal association of elements with identical subscripts may not occur, i.e. the actual element A(2,2) is probably not associated with the dummy element D(2,2). Within the subprogram, the dummy array declarator is used to reference the actual array and if any subscripts other than the rightmost one are different between the dummy and actual declarators, the same subscript does not reference the same array element in both the calling program and the subprogram. Therefore, agreement of the declared dimensions between actual and dummy arrays is usually desirable and often necessary for correct interpretation of the program.

Array declarators in subprograms may use integer variable names if they are dummy arguments and if the array is also a dummy argument. This feature is referred to as adjustable dimensions and it allows the calling program to supply dimension information that allows proper referencing of actual arrays of different dimensions. The values of the adjustable dimensions cannot be changed within the subprogram.

The standard does not allow dummy argument arrays to be declared with more array elements than are in an associated actual argument and also does not allow referencing more array elements than declared in the dummy argument array declarator. Therefore, if all elements of the actual argument are to be referenced, the dummy and actual argument array declarators must declare the same number of elements. However very few computers actually enforce either of these standard restrictions.

The following examples illustrate the associations between elements of an actual argument array A and a dummy argument array D:

```
Array Declarators:  A(2,2) and D(2,2)
Actual Argument  :  A or A(1,1)

          A(1,1) ~ D(1,1)
          A(2,1) ~ D(2,1)
          A(1,2) ~ D(1,2)
          A(2,2) ~ D(2,2)

Array Declarators:  A(3,2) and D(2,3)
Actual Argument  :  A or A(1,1)

          A(1,1) ~ D(1,1)
          A(2,1)   D(2,1)
          A(3,1) ~ D(1,2)
          A(1,2) ~ D(2,2)
          A(2,2) ~ D(1,3)
          A(3,2) ~ D(2,3)

Array Declarators:  A(2,3) and D(3)
Actual Argument  :  A(2,1)

          A(2,1) ~ D(1)
          A(3,1) ~ D(2)
          A(1,2) ~ D(3)
```

2. FUNCTION Statement:

```
FUNCTION a(b,c,d,...)
```

This statement is similar to the SUBROUTINE statement but should be used whenever the subprogram has only one result. No dummy argument should be listed for the result as it is intended that the function will be used in an arithmetic (or logical) expression and the result is simply used in evaluating the rest of the expression.

The name of the function is a and b,c,d,... represent non-subscripted dummy arguments. The name of the function indicates the type of the result by its first letter or the type of the result may be declared before the word FUNCTION, e.g., REAL FUNCTION, LOGICAL FUNCTION, etc., and it may not appear in a type-statement or any other specification statement. The type of other dummy arguments can be specified in type-statements within the subprogram. On BRLESC I, the name of the function must not end with F if it consists of more than three characters and does not begin with I, J, K, L, M, or N.

Within the FUNCTION subprogram, some statement should store a value in a variable that has the same name as the name of the function and this will be used as the result.

There must always be at least one dummy argument for FUNCTION subprograms.

```
Examples:  FUNCTION LOW(Q1,T)
           LOGICAL FUNCTION FOUND (L,V,N)
```

3. RETURN Statement:

```
RETURN
```

This statement may be used as often as desired within subprograms to indicate the point or points at which execution of the subprogram should stop and control should return to the program that is using the subprogram. It should always be used at least once in every subprogram.

4. END Statement:

```
END
```

This statement should be used at the end of all subprograms and at the end of the main program. It is not required on BRLESC I/II. All program decks on BRLESC I/II do require the very last card of the entire program deck to be a card that has an E in column 1 or an * in column 1 with "DATA" in the statement field.

For BRLESC I/II, the main program and all the subprograms must be compiled at the same time and execution automatically begins after compilation if no errors have been detected during compilation. BRLESC I has a limit of 60 subprograms used in any one program deck and BRLESC II has a limit of 255 subprograms.

5. ENTRY Statement:

ENTRY a(b,c,d,e,...)

The purpose of this statement is to allow multiple entry points within subprograms. It is not a standard FORTRAN statement, but some form of it is allowed in a number of FORTRAN IV compilers. The following description applies only to BRLESC I/II and is not completely compatible with any other computer.

The name of the entry point is a and b,c,d,e,... are the names of nonsubscripted dummy arguments. The name of the entry point a is used in a CALL statement for ENTRY statements in subroutine subprograms and is used in arithmetic expressions for ENTRY statements in function subprograms.

The dummy arguments in an ENTRY statement do not have to be the same as those in the SUBROUTINE or FUNCTION statement for the subprogram in which the ENTRY statement appears. However, a dummy argument may not appear in <u>any</u> statement (including DIMENSION) unless it has previously been declared to be a dummy argument by appearing in a SUBROUTINE, FUNCTION or ENTRY statement. The ENTRY statement must also physically precede all of the appearances of any of the dummy arguments that will actually be used in executable statements for that entry to the subprogram. (This is the only essential difference between 7090/7094 and BRLESC I/II ENTRY statements. 7090/7094 allow dummy arguments to be used both before and after the ENTRY statement.)

The name of the result in a function subprogram cannot be an entry name. Only the name appearing in the FUNCTION statement is allowed as the name of the result.

ENTRY statements are nonexecutable and normal control may pass through them without doing the initializing of the arguments for that entry.

BRLESC I/II has a limit of 100 dummy arguments and entry names in ENTRY statements within one subprogram.

CDC FORTRAN allows ENTRY statements without dummy arguments. It uses the original dummy arguments automatically with each entry.

Example: ENTRY TRY2(V,R)

6. BLOCK DATA Statement:

BLOCK DATA

This statement is used to begin a specification subprogram that allows the DATA statement to store constants into variables that are in labeled common blocks. This subprogram must not contain any executable statements. It must contain one or more COMMON statements that list all of the names that are in any of the labeled COMMON blocks that is to receive constants from a DATA statement. It is not permissible on most computers for any DATA statement to store into a blank COMMON variable; however this nonstandard feature is allowed on BRLESC I/II.

The use of BLOCK DATA subprograms is not necessary on BRLESC I/II but it should be used to maintain compatibility with other computers. BLOCKD will be used as the name of the BLOCK DATA subprogram in a BRLESC I/II dictionary.

Example:
```
BLOCK DATA
DIMENSION A(6)
LOGICAL LA
COMMON/B1/R,A/B2/V,LA
DATA LA,A/.TRUE.,6*1.0/
END
```

XV. PREDEFINED FUNCTIONS AND STATEMENT FUNCTIONS

FORTRAN subroutines are separated into two classes, (1) functions are those routines that have only one result and hence may be used in arithmetical (or logical) expressions; and (2) SUBROUTINE subprograms (See Section XIV) or other subroutines that may have more than one number as a result and may be used only by CALL statements.

Functions

There are three methods of defining a function. They are

1. Predefined functions that may be used simply by using the predefined name.
2. Statement functions.
3. FUNCTION subprograms. (See Section XIV)

Predefined Functions

Appendix A lists the predefined functions that are allowed on BRLESC I/II and most computers. Both the FORTRAN II and IV names are listed for each function and either name is allowed on BRLESC I/II. The standard name is the same as the FORTRAN IV name for those functions that are predefined by the standard.

Additional function subprograms are available from the Systems Programming Branch in the form of card decks.

Naming Functions

For FORTRAN IV and standard FORTRAN, all function names indicate the type of result in the same manner as other variable names, i.e., either the initial letter determines the type or the type is declared in a type-statement or FUNCTION statement. It is best to avoid using function names that end with F when they have more than three characters. See the description of the FUNCTION statement (Section XIV) and the description of statement functions in this section for BRLESC I/II restrictions on the use of such names. These restrictions arise because of the incompatible naming conventions between FORTRAN II and IV and the desire to allow most FORTRAN II and IV programs to execute properly on BRLESC I/II.

For FORTRAN II, predefined function (and statement function) names must always end with F (a total of seven characters are allowed) and must begin with X only if the result is an integer. BRLESC II will also use statement function names that begin with I-N as of type integer. Variables must never be given a name that is the same as any of the function or subroutine names either with or without the terminal F. For BRLESC I/II, the terminal F is not necessary when the initial letter of the predefined function name indicates the proper type of result but is necessary in both the definition and use of arithmetic statement functions whenever it appears either place.

Use of Functions

Any of the three types of functions may be used in an arithmetic expression by writing its name in front of a pair of parentheses that enclose the list of arguments. The arguments must correspond in type, order, and number to the dummy arguments used in defining the function. Successive arguments are separated by commas and they may be arithmetic expressions.

For BRLESC I/II, any function may also be used in a CALL statement by adding one extra actual argument that specifies where to store the result.

Statement Functions

Statement functions are functions that can be and are defined by one statement at the beginning of a main program or subprogram. The name of the function followed by the dummy arguments enclosed in parentheses are written to the left of the = symbol. The expression that describes the function in terms of the dummy arguments is written to the right of the = symbol. The dummy arguments cannot be subscripted. Any variable used in the expression that is not a dummy argument will be identical to the variable of the same name in the main program or subprogram in which the statement is contained. A statement function definition normally can be used only in the program or subprogram in which it is located, however BRLESC I allows them to be used anywhere within the complete program.

A statement function may use any of the other types of functions and may also use other previously defined statement functions. All statement functions must precede the first statement that gets executed in the program or subprogram.

If the statement function name does not indicate the proper type of result, then its name must appear in a type-statement. When a statement function name appears in a type-statement on BRLESC I, it must also be put in an EXTERNAL statement that appears after the type-statement and this pair of statements must appear in every subprogram that uses the statement function. On BRLESC I, the name of a function must not end with F if it consists of more than three characters and does not begin with I-N. On BRLESC I/II, statement function names that have more than three characters, end with F, and begin with X will be used as integer unless declared a different type in a type-statement. On BRLESC I, statement function names that have more than three characters, end with F, and begin with I-N will be used as real unless declared a different type in a type-statement.

The dummy argument names must indicate the same type of arithmetic that is required when the function is actually used. When the initial letter of a dumr. argument does not indicate the proper type, it may appear in a type-statement before the statement function. When this is done, the BRLESC I dictionary will not have the variable marked as a dummy argument, but the program will be correct.

A statement function may be any kind of assignment statement, i.e., it may be arithmetic, logical or a nonstandard masking statement.

Example of defining an arithmetic statement function:

```
FUN(A,B,C) = A**2 - SIN(B*C)+C
```

Example of using this arithmetic statement function:

```
T = Q + FUN(X,S + EXP(V**2),14.)
```

XVI. PREDEFINED SUBROUTINES

A subroutine may be predefined and supplied by the compiler or it may be defined by a subroutine subprogram. (See Section XIV.) Subroutines may be given any valid name (no restrictions on the first or last letter) and may only be used by a CALL statement.

There are no subroutines predefined by th^ standard. The following subroutines are predefined in BRLESC I/II FORTRAN:

SETMSI	(j)	Set minus sign for input.
SETPSI	(j)	Set plus sign for input. (Not necessary, anything not minus is plus.)
SETMSO	(j)	Set minus sign for output.
SETPSO	(j)	Set plus sign for output.

where j is an integer constant:

0 means blank.
1 means y (12) punch.
2 means x (11) punch.
3 means x or y punch.

SEXAPR(a,b)	Sexadecimal print from the address of a to the address of b.
BINPUT	Goes to binary input routine after saving a return jump instruction (in 073 on BRLESC I and 007 on BRLESC II).
POWERS(a,b,c)	Computes c = a**b where b may be integer or real.
SINCOS(a,b,c)	Computes b = SIN(a) and c = COS(a).
CHTAPE(u,t)	Change FORTRAN I/O unit u to use BRLESC I/II tape switch t. (u and t must be integers.) This should only be executed before any I/O has occurred on unit u.
MATMPY(a,b,c,i,j,k,im,jm,km)	Multiply matrices; c(i,k) = a(i,j) * b(j,k) where im,jm,and km must be the declared maximum row dimensions of a,b, and c respectively.
PLTCCA	Plot routines for Calcomp plotter.
PLTCCB	A separate ARDC Technical Note describes the
PLTCCD	argument lists and provides detailed informa-
PLTCCE	tion on using these plotting routines.
PLTCCP	
PLTCCS	
PLTCCT	
FIXSCA	
CONSCA	

RDCLK(r)	Read clock into r (alphanumeric characters).
STCLKS(r1,r2,d)	Subtract clock readings (r2-r1) and store difference in minutes in d as a real number. If r1 is a blank argument, the start-time (of compilation) is used. If r2 is a blank argument, the current time is used.
CVCLK(r,m)	Convert clock reading r into minutes since previous midnight and store in m as a real number. If r is a blank argument, the current clock reading is used.
CKCLK(t,s) or CKCLK(t)	If total (compile + execute) BRLESC I/II time is greater than t real minutes, do statement s next. (Statement number s must have an S after it.) When s is omitted, it is equivalent to specifying a STOP statement.
UNPACK(a,b,n)	Unpack n characters from ten characters per word beginning at a to one right adjusted character per word beginning at b. On BRLESC I, a and b must have subscripts if they are arrays. If n = 0, the subroutine does nothing.
PACK(a,b,n)	Pack n characters from one right adjusted character per word beginning at a into ten characters per word beginning at b. On BRLESC I, a and b must have subscripts if they are arrays. If n = 0, the subroutine does nothing.
SETREB	Sets I/O routine to also allow a blank column to start an exponent in input numbers.
SETCWD(i)	Sets so the DECODE statements will decode i characters per word. (integer i ≤ 10)
SETCWE(i)	Sets so the ENCODE statements will encode i characters per word. (integer i ≤ 10)

SKIPFILE(u,n) or SKIPFILE(u)	Moves FORTRAN tape u forward n file marks if it is 1/2" tape. When n is omitted or if the tape is 1", the tape moves forward only to the next file mark. On BRLESC I, n = 0 is the same as n = 1. Only input tapes may be specified and not tape switch 6 which is usually FORTRAN unit 5. The next READ (or WRITE) statement will begin with the information after the file mark except BRLESC I will erase the file mark if the next statement is WRITE.
BACKFILE(u,n) or BACKFILE(u)	Moves FORTRAN tape u backward n file marks if it is 1/2" tape. When n is omitted or if the tape is 1", the tape moves backward only to the next file mark. On BRLESC I, n = 0 is the same as n = 1. Either input or output tapes can be specified, but not BRLESC tape switches 6 or 8. The next READ statement will begin with the information after the file mark, the next WRITE will erase the file mark except BRLESC I erases the previous block of information too. (Note: BRLESC II properly does BACKFILE & SKIPFILE including the situation when it has read a block ahead. BRLESC I does not properly start writing immediately after doing BACKFILE or SKIPFILE.)
FORTRAN	BRLESC II only. Calls the FORTRAN compiler to begin compilation of another program, which may be either a FORTRAN or FORAST program. A PROB card (PROB in cols. 7-10) must immediately precede the next program. This subroutine allows two or more programs to be run consecutively, but submitted as one program deck.

Additional predefined subroutines may be added in the future and additional subroutines in the form of card decks are available from the Systems Programming Branch.

XVII. FORTRAN PROGRAM CARDS

BRLESC I/II use the standard card format for punching FORTRAN programs.

Columns:

1 - 5	Statement number (integer).
6	Continuation Card if not zero or blank.
7 - 72	One FORTRAN statement. (BRLESC I/II allow more than one.)
73 - 80	Identification.

The statement number must be a decimal integer. Leading zeros and all blank columns are ignored. On BRLESC I/II, if a statement number is the same as the last nonblank statement number field, it is ignored.

Column 1 is also used to indicate special types of cards. The following list shows the special characters that indicate special cards:

C	Comment card. Columns 2-80 may be used for comments.
*	BRLESC I/II control card.
B	Boolean statement card. (FORTRAN II, not allowed on BRLESC II)
D	Double Precision statement card. (FORTRAN II, not allowed on BRLESC II)
I	Complex Arithmetic statement card. (Not allowed on BRLESC I/II)
F	Used to specify names of subroutines and functions used as arguments. (FORTRAN II, not allowed on BRLESC II)
$	BRLESC I assembly order cards and BRLESC I/II MAXT and MAXO control cards.

-	BRLESC II assembly order cards. Ignored by BRLESC I.
7-8	End-file signal on 7090/7094, control card signal on 1108, ignored on BRLESC I/II.
E	BRLESC I/II, is last card of program deck.
X	BRLESC I will use FORTRAN statements from this card and BRLESC II will ignore this card.
Y	BRLESC I will ignore this card and BRLESC II will use FORTRAN statements from this card.
/	Job control cards on 360 computers, ignored on BRLESC I/II.

All of these special column 1 indicators are nonstandard except the C in column 1 for comment cards.

Column 6 is used to mark cards that are a continuation of the previous card. It is used as a continuation if column 6 contains any character other than zero or blank except on the initial BRLESC I/II identification control card and all comment cards. BRLESC I/II does not limit the number of continuation cards allowed for one statement but the standard prescribes a limit of nineteen.

Columns 7-72 contains information, one or more statements, com-
coments, control information, etc. depending on the type of cards as indicated by Column 1. BRLESC I/II will allow more than one statement per card if the symbol $ is used to separate the statements. A special program is available for compacting and repunching a FORTRAN program so that it will have more than one statement per card and another program is available for repunching the compacted version with one statement per card.

Columns 73-80 are ignored and may contain any desired identification, card number, etc.

Blank columns are ignored except when they are in hollerith information in a FORMAT statement or within alphanumeric constants.

Blank cards will be ignored on BRLESC I/II, but the standard does not permit "empty statements" which are usually caused by blank cards.

F Cards (FORTRAN II, not BRLESC II)

If the name of a subroutine or function, either predefined or defined by a subprogram, is used as an argument for another subroutine or function, its name, usually without the terminal F, must appear on a card with an F in Column 1 or in a standard EXTERNAL statement. The F card or EXTERNAL statement must be in the program or subprogram that uses the subroutine or function as an argument and the F card may be anywhere within that program or subprogram.

The names of the subroutines and functions are to start in or beyond Column 7 and are separated by commas.

```
Example:      F     SIN, EXP, FUN3, ATAN
```

On BRLESC I, the terminal F is to be omitted from those function names that have an initial letter that indicates the proper type of result according to the I-N rules. It must be retained on those names that do not indicate the proper type of result, e.g., LOGF, MAXOF. This same rule for the terminal F applies where the name is used as an argument for a subprogram. When programming a subprogram to accept a function name as an argument, the dummy argument should end with F only if the initial character does not indicate the proper type of result. If a final F is used with at least three other characters, then the result type is integer only if the name begins with X.

For FORTRAN IV and standard FORTRAN, the EXTERNAL statement replaces the F card and serves the same purpose.

B Cards (FORTRAN II, not BRLESC II)

Cards with B in column 1 contain boolean statements where + means the inclusive or operation, * means the and operation, and - means the not operation. Integer constants are octal constants on these cards.

BRLESC I performs these boolean operations on the rightmost 65 bits of a word and sets the other 3 bits to zero.

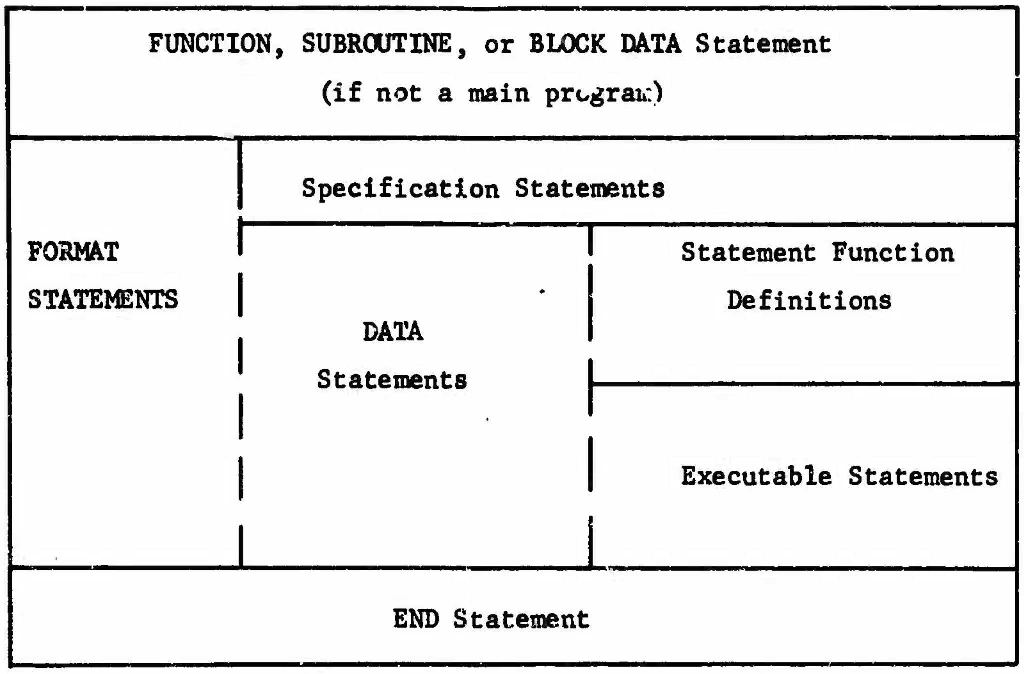
D Cards (FORTRAN II, not BRLESC II)

Cards with D in column 1 indicate that the statements on such cards are to be executed with double precision arithmetic. BRLESC I allows such cards, but executes the statements in single precision (about 16 decimal digits).

Statement Arrangement

On BRLESC I/II, all of the statements for the main program must physically be before the subprograms. The subprograms may be in any order, but no subprogram may contain any statements that are a part of another subprogram. BRLESC I allows a total of 60 subprograms, BRLESC II allows 255 subprograms.

Within each main program or subprogram, there are a few restrictions on the physical arrangement of statements. The following diagram shows that (1) FORMAT statements may appear essentially anywhere (2) all specification statements must appear before all DATA statements, statement function definition statements, and executable statements (3) DATA statements may appear interspersed with statement functions and executable statements (4) all statement function definitions must appear before all executable statements (5) the first statement must be a FUNCTION, SUBROUTINE, or BLOCK DATA statement if it is not a main program and (6) the last statement must be an END statement. This arrangement is standard and is necessary on BRLESC II.



Physical Arrangement of Statements

BRLESC I has a further restriction that when the same name appears in DIMENSION, COMMON and EQUIVALENCE statements, or in any two of these statements, the statements involved must appear in the order of DIMENSION, COMMON, and then EQUIVALENCE within the specification statements. Actually, BRLESC I does not require that all of the specification statements appear before the statement functions and executable statements. It does require that the first appearance of an array name be its array declaration and any type-statement must appear before the names appear in an arithmetic expression.

XVIII. BRLESC I/II CONTROL CARDS AND DICTIONARY PRINTING

The use of certain control cards are allowed to affect the compilation of FORTRAN programs. Most of these apply to BRLESC I/II only, although some are also used on other computers. All of the BRLESC I/II control cards are marked with an * in Column 1 with the control information starting in or after Column 7.

*   The first card of a program that has an * in Column 1 is used as identification and is printed in front of the normal output. Columns 2-80 may be used. (On all other cards with * in Column 1, only Columns 7-72 may be used.) The first thing after the * should be the official problem number followed by a blank column or comma and this should not extend beyond column 20. This card should also contain the programmer's name, phone number, and building number.

*   SETSSW i $\begin{Bmatrix}\text{UP}\\ \text{DOWN}\end{Bmatrix}$

This control card allows sense switch i to be "preset" either UP or DOWN. By using this control card, the operator can be relieved of actually setting the sense switches.

*   PRTOPU

This control card causes the compiler to translate all following PRINT statements as though they were PUNCH statements. (Allows card output instead of tape.) Note that vertical space control characters that are explicitly in a FORMAT statement will be printed when PUNCH statement output is listed.

*   RTTORC

This control card causes the compiler to translate all following READ INPUT TAPE, INPUT or READ(t,f) statements as though they were READ statements. (Use card input instead of tape.)

*   WTTOPU

This control card causes the compiler to translate all following WRITE OUTPUT TAPE, OUTPUT or WRITE(t,f) statements as though they were PUNCH statements.

*   WTTOPR

This control card causes the compiler to translate all following WRITE OUTPUT TAPE, OUTPUT or WRITE(t,f) statements as though they were PRINT statements.

*   MVPRTO(sexa. add.)

This control card may be used (usually before the first executable statement) to move the program to the specified sexadecimal address.

*   SUBR(sub. name = sexa. add.)

This control card allows any specified predefined subroutine (or function) to be stored at the specified sexadecimal address.

The above two control cards are needed when the space allowed for storing predefined functions and subroutines is exceeded as indicated by error print 73. This usually occurs only when the plotting subroutine PLTCCB is used since it is very large. In that case, the following two control cards should be used before the first executable statement in the main program:

```
*     MVPRTO(01K00)
*     SUBR(PLTCCB = 01400)
```

*   BIG CLEAR

This control card will cause BRLESC II to initialize all uninitialized variables to a large number instead of the normal zero. It may appear essentially anywhere within a program and uses the sexadecimal constant OLZ1LLLL to initialize variables. Standard programs must not assume that variables have a value of zero initially and this control card assists in determining if all variables are given values by the program before they are used to compute other quantities. BRLESC I will ignore this control card.

* /0 = 0

This control card causes BRLESC II to print an error print each time a real division by zero is attempted. It also causes the result of the division to be zero and allows the program to continue running. It causes a subroutine to be used to perform all real divisions and therefore increases the length and execution time of the program. The form of the error print is:

```
/0 = 0   AT   a   b/d
```

where a is one more than the sexadecimal address at which the division by zero occurred, b is the numerator and d is the divisor which should be zero.

* COMPILE TAPE TO

* COMPILE TAPE THRU

* SKIP TAPE THRU

See Section XXIII for explanation of these three control cards that are used when compiling from tape 12.

* MEMORY i1,i2,i3

This control card permits a limited reassignment of storage space within the BRLESC II compiler. It is ignored on BRLESC I. It may allow a program to be run that previously caused compile error prints of number 19, 63, or 82. The i1, i2, and i3 are integer constants that normally have values of 3072, 4096 and 2048 respectively.

i1	Maximum number of names allowed in the dictionary.

i2 Maximum number of words in the second part of the dictionary. One word is required for each name plus at least 10% for other storage assignments.

i3 Maximum number of words reserved for keeping information that was in DATA statements. One word is required for each constant and about one word for each name.

The following restrictions must be satisfied:

$$i1 + i2 < \text{Memory Size} - 25000$$

$$i3 < 3500$$

where "Memory Size" is 32768, 49152, 65536, or 81920 depending on the amount of memory requested on the program cover card.

* LIST

* SYMBOL TABLE

Either of these causes the storage dictionary to be printed. The asterisk in Column 1 is not required on the LIST card.

The dictionary is printed with names of variables arranged in alphabetical order within each subprogram. Function and subroutine names will be preceded by two asterisks and will appear at the very beginning of the dictionary. BRLESC II changes FORTRAN II predefined function names to their FORTRAN IV and standard equivalent before inserting them in the dictionary and inserts a blank character at the beginning of the names of standard "intrinsic functions". On BRLESC I, main program names will be preceded only by two blanks and subprogram names will be preceded by one character and one asterisk or period. The character preceding each subprogram name will be 1,2,...,9,A,B,...T corresponding to the sequence in which the subprograms appeared in the program deck. The name of each subprogram will appear on a separate line before the dictionary for that subprogram. If more than 30 subprograms are used on BRLESC I, some dictionaries for two subprograms will be mixed together with both subprogram names preceding that section of the complete dictionary. When this occurs, those names preceded with an asterisk are from the subprogram whose name appears on the left side of the subprogram name card and those names preceded with a period are from the subprogram whose name appears on the right. BRLESC II prints the dictionary for every subprogram separately and does not precede each name with any special characters. It prints the number of each subprogram on the line that contains the name of the subprogram. The subprogram dictionaries appear in the same physical order that the subprograms appear in the program.

Following each name will be the sexadecimal memory address that has been assigned to the name. Following this address, any of the following letters may appear:

A	indicates an array name.
I	indicates an integer variable.
L	indicates a logical variable.
C	indicates the name was in a COMMON statement.
E	indicates the name was in an EQUIVALENCE statement.
U	indicates the name was used only once. (BRLESC II precedes the U with the absolute card number on which the symbol appeared, if it was not in a specification statement.
R	indicates a name was in a REAL statement. (BRLESC II)
D	indicates a double precision variable. (BRLESC II)
'	Appears after a statement number when that statement begins in the right half of a word. (BRLESC II)

Statement numbers are printed at the right end of the six character name position and therefore always precede the names of the variables in any subprogram. The BRLESC I compiler usually adds a few names to the dictionary to indicate temporary storage and special subroutines. The name $SUBS. is printed usually at the end of the dictionary to indicate the storage required by the predefined subroutines. The predefined subroutines extend from this address down through 0103L and includes all of the input/output routines and subroutines. The $NOS. name is printed usually as the next to last name in the dictionary and indicates the length of the "constant pool". This storage, from OSO (OFO on BRLESC II) down to but not including the address printed after $NOS., is used to store the constants and the "array words" (BRLESC II does not use array words) required by the program. The $LAST entry printed with the dictionary indicates the largest address used by the program with the possible exception of some tape buffers at the end of the memory.

For array names, the address printed in the dictionary is the initial address of the array.

The names of all the common variables used within a subprogram may not appear in the dictionary for that subprogram. When the COMMON statements of a subprogram are processed, a check is made to determine if the names and required storage are the same as those for the main program. All of the names up to the point of the first disagreement in name or storage are deleted from the BRLESC I subprogram dictionary. BRLESC II keeps all the names if there is any disagreement and deletes all the names when they agree. If the subprogram common statements are identical to the main common statements, then the words MAIN COMMON are printed preceding the subprogram dictionary. If the first common name of the subprogram

disagrees with the first common name of the main program, then the check and deletion explained above is made with the common statements of the previous subprogram.
If the subprogram common statements are identical to the previous subprogram common statements, then the name of the previous subprogram followed by "COMMON" is printed preceding the subprogram dictionary on BRLESC I and BRLESC II prints "PREVIOUS COMMON".

Names in blank common are assigned last, so the last name in the blank common assignment within the subprogram that has the most total (blank and labeled) common storage will mark the end of all the storage used by the program. The instructions for the program and all the subprograms are stored first, then all the variables not in common and not assigned before the predefined subroutines are assigned storage immediately after the instructions and this is followed by those variables in labeled common, with the blank common assigned last.

*　　LIST8

*　　LIST (S.CODE)

Either of these control cards causes the dictionary and the sexadecimal code for the entire program to be printed. Four instructions are printed on a line with the address of the first one printed at the beginning of the line. The * in Column 1 of LIST (S.CODE) may be omitted unless LIST is the name of an array.

*　　LIST (B.CODE)

On BRLESC I, this control card causes the entire program and the subroutines it uses to be punched on binary cards with absolute addresses. To use this binary deck to run the program, it must be preceded by a binary input routine and followed by the standard set of FORTRAN input/output routines and a jump to 073. The use of binary decks is not recommended because BRLESC I can probably compile the FORTRAN program from tape faster than reading the equivalent binary cards. The * in column 1 may be omitted unless LIST is the name of an array. On BRLESC II, this card is the same as the LIST (S.CODE) control card.

*   LIST (START)

This control card may be inserted anywhere after the identification card to cause the following source program cards to be printed on the normal output unit. This card is ignored if inserted before the identification card. It is effective until a LIST (STOP) control card or the end of the program is encountered. It does not cause a dictionary to be printed. The * in column 1 may be omitted if LIST is not an array name. Each LIST (START) card causes the following statements to start printing at the beginning of a new page.

*   LIST (STOP)

This card causes the compiler to stop printing the source program cards. It only has meaning when it has been preceded with a LIST (START) control card. Note that pairs of LIST (START) and LIST (STOP) cards can be used to print any selected portions of a program although it is not required that they appear in pairs. This card does not cause a dictionary to be printed. The * in column 1 may be omitted if LIST is not an array name.

XIX. BRLESC I/II ASSEMBLY ORDERS

BRLESC I Assembly Language

BRLESC I allows BRLESC I assembly orders to be written on cards that have a $ in column 1. The same general form as used in FORAST is allowed, but not the special processing of addresses, formulas, etc. FORTRAN statements must not be put on the same card with assembly orders, but more than one assembly order may be put in columns 7-72 by separating them with a $ symbol. The general form of each order is like FORAST, i.e., OT(A)B)C$ where "(" after ")" is optional, the last ")" and the last $ are optional and less than three addresses is permitted. Comments may follow $$.

For BRLESC I assembly orders:

Col. 1	$
Cols. 2-5	Blank, statement number or symbolic name is allowed. Successive duplicate numbers or names are not allowed.
Col. 6	Normal FORTRAN continuation column. ($ in col. 1 not required on continuation cards.)
Cols. 7-72	One or more BRLESC I assembly orders.
Cols. 73-80	Identification only.

The following symbolic order types are the only ones allowed:

A	B	SET	J-	JC	MI
S	CB	SI	TAPE	NOP	IM
M	CEQ	INC	CARD	RSW	RCL
D	CNB	II	ZERO	MMF	
C	CNEQ	LP	SIJ	LPI	
SQRT	PMA	J	IIJ	MMB	
SHX	IT	JS	EA	JNA	
TP	HALT	J+	JA	JNC	

See Appendix C for a brief description of these BRLESC I orders.

Any of the symbolic parameters X, F, A or +, V, and R may be written after any arithmetic order type (i.e., A, S, M, D, C, SQRT, SH, and PM) without punctuation and in any sequence. Note that SH must always have an X parameter and PM must always have an A parameter. The C order type may also be written C-. These parameters have the following meanings:

X	means fixed point fractional arithmetic.
F	means floating point arithmetic.
A or +	means to accumulate the result in C.
V	means use absolute value of both operands.
R	means use the "R register".

A decimal parameter is also allowed. All arithmetic orders are floating point unless the X parameter is used.

A GOTO statement with one address is allowed but none of the other general FORAST statements are allowed.

No assembly order may have more than three addresses including SET and INC and they cannot include a GOTO. However any of the orders that set or increase index registers may be written with an = like FORAST allows, e.g., SET(I=3)$. Index registers cannot be increased by a negative amount.

A comma is used to indicate indexing in the same manner as FORAST. Constant increments (and decrements on symbolic addresses) are allowed. All index addresses must be absolute, decimal or sexadecimal. Addresses 13-27(0J-01S) and 48-55(030-037) are not normally used in FORTRAN compiled programs. Dummy arguments must never be indexed and variables that might be assigned to a "large address" (addresses greater than sexadecimal 03LLL) must not be indexed. Large addresses may be used anywhere as long as they are not indexed. (BRLESC I allows large addressing by using indexing automatically on all large addresses.)

FORTRAN array subscripting is not allowed in any BRLESC I assembly orders. When an array name is used by itself, it references the "array word", not the first element of the array.

Decimal addresses are allowed and sexadecimal addresses must have a leading zero. Statement numbers used for an address must either be preceded by "S/" or followed by an S. Decimal and hollerith constants must be preceded by an * and may be either a FORTRAN integer, floating point, or hollerith constant. Fixed point fractions are not allowed. Sexadecimal constants may be written following a leading "/" and may use M,A and Z, i.e., M = LLLLL, A = 00000, and Z represents enough zeros to fill out a word of 17 sexadecimals.

There is no special processing of any addresses like there is in FORAST for β of SH,JA,JNA,JC,JNC and γ of I/O orders. These addresses should normally be written in sexadecimal.

Names of subroutines may be used as an address only by preceding each one with "F/" or including each one in an EXTERNAL statement or on an "F card".

Decimal increments and decrements are allowed on symbolic addresses and may be written either before or after the index name, e.g., A+2,14 or A,14+2. SELF is also allowed to refer to an order's own address.

Symbolic names may be assigned absolute addresses by using a SYN statement on a $ card. SYN may be followed by any number of pairs of parentheses that enclose one symbolic address and one absolute address separated by "=". An example is:

```
$ SYN(A=080)BT=942)(OLOO=Z1)$
```

Note that "("after")" is optional and that $ at the end indicates that the rest of this card will not be used.

A group of sexadecimal or decimal constants may be stored by using a SEXA or DEC statement on a $ card. Any number of constants may be put on one card but no other statements or assembly orders are allowed on the same card. Constants are separated by parentheses with "(" after ")" being optional. For sexadecimal constants, Z indicates a string of zeros, A a string of five sexadecimal zeros and M a string of five sexadecimal L's. Any legal FORTRAN decimal constant may be written on a DEC card.

```
Examples:  $   SEXA(L)08Z)*10KZ82)4A$
           $   DEC(14)3.)6.1E-3)$
```

Up to six alphanumeric (hollerith) constants may be stored by using ALFN in columns 7-10 and $ in column 1. Columns 11-20 are always stored as one constant, one additional word is stored for each ten columns until one is blank (it is not stored) or until columns 61-70 have been stored.

```
Example:   $   ALFN THIS IS ALPHABETIC INFO.
```

Comment lines with COMM in columns 7-10 and $ in column 1 are allowed and should be used to provide BRLESC I/II operators with information about the program that is useful to them, e.g., information about the tapes required.

```
Example:   COMM TAPE INPUT ON UNIT 1, LABELED 4A7.
```

Some examples of BRLESC I assembly orders are:

```
           $   AV(F)*7.1)T1$ SHX(Q,2)0286)0$
           $   TAPE(SS2B)360)017 $ SET(15=0)3=1$
           $   INC(2=2+1)21=21+4)$ CNEQ(W)/2A)42S$
           $   B12(R-1,4))TT$ TP10(I2)/M)SELF+2$
           $   JS(A)A+50)(F/SEXAPR)$ GOTO(TEST)$
           $   029(07000)S,24-1)II$ MMF(1,6)300)08000$
```

BRLESC II Assembly Language

BRLESC II allows BRLESC II assembly orders to appear within FORTRAN programs on cards that have a - (minus sign) in column one. Each card that contains any assembly language must have the "-" in column one and such cards must not contain any FORTRAN statements.

Each assembly order has an order type followed by one optional address enclosed in parentheses and a $ character indicates the end of an order. Thus the general form is "order type(address)$" where the address and its enclosing parentheses may be omitted, the ")" after the address may be omitted, and the $ may be omitted after the last order on a card that is not continued on the next card. Comments may appear after "$$".

For BRLESC II assembly order cards:

Col. 1	- (Minus sign)
Cols. 2-5	Blank, statement number or symbolic name that begins with a letter.
Col. 6	Normal FORTRAN continuation column. (- in col. 1 not required on continuation cards.)
Cols. 7-72	One or more assembly orders.
Cols. 73-80	Identification only.

Location Field (Cols. 2-5):

BRLESC II allows 2, 3 or 4 orders per word and allows jumps to either the left half of a word or the right half. When a statement number appears in columns 2-5, the first order on that card will start at either the left half or the right half of a word. When a symbolic name appears in cols. 2-5, the first order on that card will start at the left half of a word except when the name begins with R; then the first order will start at the right half of a word. A symbolic name used as a location of assembly orders must not have previously been assigned memory space. Therefore, the location must be its first appearance or else it must have appeared in some specification statement or appeared only as a primary address in assembly orders.

Successive duplicate names or statement numbers are allowed, only the first one is used.

Symbolic Order Types:

Appendix D lists the acceptable symbolic order types and a brief explanation of each order. Note that the jump orders that contain a '(prime) character will always jump to the right half of a word. The other jump orders jump to the left half of a word, but the compiler changes them to the prime orders if the specified symbolic address has been assigned to an order that begins at the right half. Therefore the jumps without the primes should normally be used. All absolute (dec. or sexa.) addresses are assumed to be locations of left orders when used as addresses in jump orders without primes. Sexadecimal order types are also allowed, e.g. ON4.

Addresses:

An assembly order address may consist of three parts; a primary address, an index address, and a decimal increment that may be positive or negative.

The assembly order primary address may be any of the following types:

(1) Symbolic. Begins with a letter.

(2) Statement number. Must have a trailing S; e.g., 13S.

(3) Sexadecimal. Begins with zero.

(4) Decimal. All decimal digits.

(5) Decimal Constant. Preceded with "*", may be integer, real or hollerith (H, R or prime). May have a sign.

(6) Sexadecimal Constant. Preceded with "/"; A, M and Z characters may be used.

(7) Subroutine or Function Name. Must be preceded with "F/".

All addresses except constants (types 5 and 6 above) can be indexed by writing a comma after the address followed by an index address. The index address must be symbolic, sexadecimal, or decimal. The effective address is strictly a sum of the address assigned to the primary address and the contents of the index address plus any increment. Thus if A is an array; A, I addresses A(1) when I=0 and A(2) when I=1, etc. Note that only "one dimensional" indexing is allowed and normal FORTRAN subscripting is not allowed although the name of an array can be used as the address of the first element of that array.

If the primary address is not the name of a dummy argument, a decimal increment can be used in addition to indexing. The decimal increment may be written either before or after the index address. Some examples of addresses with increments are: (B + 2) (B - 4) (Q + 17,J) (TV4, K3 + 6) (, N + 2) (-1). Dummy arguments may be indexed but cannot have an increment. An increment can appear beyond a comma only if it follows an index address.

IOS Address:

The primary address of the IOS order gets special processing to allow symbolic specification of various I/O operations. The table below lists all of the special symbolic names that can be written. More than one can be written if they are separated by minus signs. Normal indexing is allowed but an increment is allowed after an index address only. The address may be specified completely as one sexadecimal address. It may be one decimal address only if it is less than 256.

Special IOS address symbols:

1. Unit Selection:

   TAPE (optional)
   CARD
   DISC or DISK

PRINT

PUNCH

Decimal integer to select tape unit.

2. Read and Write Selection:

R (optional)	60 or 64 bits per word.
W	60 or 64 bits per word.
R72	72 bits per word.
W72	72 bits per word.

3. Special Tape Operations:

MF	Move forward by blocks.
MB	Move backward by blocks.
MFMF	Move forward by file marks.
MFMB	Move backward by file marks.
WFM	Write file mark.
REW	Rewind.
UNLOAD	Rewind and unload.

4. Character Size Selection:

C6 (optional)	Six bit characters.
C8	Eight bit characters.

5. Parity Selection:

OP (optional)	Odd parity.
EP	Even parity.
IP	Ignore parity.

Each symbol marked "optional" is used if no conflicting symbol is specified, e.g. tape is the type of unit selected if CARD or DISC, etc. is not specified. The above symbols may appear in any sequence within the primary address.

Care must be exercised in manipulating a tape with assembly orders if the same tape is used in FORTRAN statements. If possible, normal FORTRAN statements should be used. In particular, the FORTRAN formatted tape input routine reads the next block on a tape as soon as the last line of the previous block has been read by the FORTRAN program. If this has just happened on FORTRAN tape i, then the 4 tag bit (bit 67) of OFNO,i is a one, otherwise it is zero. Examples of IOS orders:

```
IOS  (TAPE-MB-4)
IOS  (DISK-W)
IOS  (WFM-3)
IOS  (R72-C8-7-EP)
IOS  (RFW,NTAPE +1)
IOS  (02409)
```

Address assigning and order length:

The BRLESC II FORTRAN compiler assigns nonarray variables to short memory addresses at their first appearance in an executable statement if they did not appear in a specification statement. However, the appearance of a name as a primary address in an assembly order does not cause its assignment to a short address. All unassigned primary addresses will cause long (half word) orders to be generated. However the appearance of an unassigned name as an index address will cause its assignment to a short address (if one is available). The length of each order is determined when it is first encountered and it is made short when possible. Unusable portions of words are filled in with OFF orders (nonindexable no operation orders). Care must be exercised when making any assumptions about either the length of an order or its position within a word. In particular, blank addresses that are to be extracted into with E or E' orders should be written as 0100 to insure the use of a half word order.

MAXO and MAXT cards:

Normal MAXO and MAXT cards have a $ in column one so that the same card will work in either FORAST or FORTRAN on both BRLESC I and II. However BRLESC II FORTRAN will accept such cards with - (minus sign) in column one while BRLESC I will ignore all cards with a minus sign in column one. Thus it is possible to specify different maximum times for the two computers by inserting the - MAXT card before the $ MAXT card since each machine will actually use only the first MAXT specified. Each machine uses the last MAXO card, so the - MAXO should be after the $ MAXO if different line limits are desired. BRLESC II does not allow any other assembly orders on the same card with MAXO or MAXT.

SYN cards:

Symbolic names may be assigned absolute addresses by using a SYN statement on a card with the minus sign in column one. SYN may be followed by any number of pairs of parentheses that enclose one symbolic address and one absolute address separated by an "=" symbol. The "(" after ")" and the final ")" are optional. A $ character marks the end of the SYN statement and the rest of the card cannot be used for anything except comments.

Example:

```
-      SYN (T = 06) 4 = W) TA = 0FNO)$
```

SEXA and DEC cards:

A group of sexadecimal or decimal constants may be consecutively stored by using a SEXA or DEC statement on a card with the minus sign in column one. The first constant will be stored at the location specified in columns 2-5 of the card and the others will follow consecutively. Constants must be separated by parentheses with "(" after ")" and the last ")" being optional. No assembly orders or any other statement can appear on the same card. Comments may follow a $ at the end of the constants.

In sexadecimal constants, A indicates five sexadecimal zeros, M indicates five sexadecimal L's, and one Z indicates enough zeros to fill out a word of 17 sexadecimals. Constants of less than 17 characters and no Z are right adjusted, i.e., (4) is the same as (Z4).

Any legal FORTRAN decimal constant may appear on a DEC card. The constants may have signs.

Examples:

```
-SE  SEXA  (KZ8)  (LNAM)N4Z)28$
-V   DEC  (19)14.26)-6.9E-4) + 18)$
```

ALFN cards:

One to six alphanumeric (hollerith) constants may be stored consecutively by using an ALFN statement on a card that has a minus sign in column one. "ALFN" must appear in columns 7-10 and columns 11-20 are always stored as one alphanumeric constant. Each following group of ten columns (21-30, 31-40, etc.) is stored until one is all blank or until cols. 61-70 have been stored. Such cards cannot be continued on the next card by using column 6.

Example:

```
-    ALFN ABCDEFGHIJKLM
```

Examples of BRLESC II Assembly Order Cards:

```
-12   F+(A)$ F(+)(B,J)$ FM(T)$ U(114S)$
-162  L+(LW)$ ANDN(/LZ)$ CZ(147S)$ $ COMMENT
-NEW  1(+)M(J)$ L+(*1)$ IORM(V,J-1)$
-RAN  LR(0100)$ + (V3)$ LSD(24)$ LM(T4)$
-     F+(AB)$ FXA(*-5.213E2)$ F/ (TM,K1)$ FM(018,X)$ U(JOB1)
```

XX. MAXIMUM TIME AND OUTPUT SPECIFICATIONS

BRLESC I/II allow the programmer to control the maximum amount of time a program will be allowed to run and the maximum amount of output it will be allowed to print. If the programmer does not specify these maximums, BRLESC I/II will set them at five minutes and 1200 lines. Whenever either one of the specified maximums is exceeded, BRLESC I/II will stop execution of the program after the appropriate error print.

Maximum Time

The maximum time specification is of the form:

```
$     MAXT(integer number)MINS.
```

where the initial $ is in column 1 and the rest of the specification is in columns 7-72. "MINS." may be replaced with "HRS." or "SECS." to specify hours or seconds instead of minutes. Note that fractions of time units are not allowed.

The time begins when this card is encountered by the compiler and hence some compilation time must be included when estimating the maximum time.

If the statement number 98765 has been used in the main program, BRLESC I jumps to that statement when the maximum time has been exceeded. If 98765 has not been used as a statement number, BRLESC I gives the following error print:

```
EXCEEDED MAXT. I1= s  OCTAL AR.REFS.= a b c CLK= cr
```

where s is the octal contents of index register 1, which is usually the address at which the last subroutine, function, or I/O routine was referenced; a,b, and c are the octal contents of indexes 10,11 and 12 respectively which are the last array addresses referenced, and cr is the clock reading at the time of the error print. This clock reading contains six digits, two each for hours, minutes and hundredths of minutes. This same error print is obtained if BRLESC I stops for some reason during execution of a FORTRAN program and the clock reading can be compared with the initial time to determine how long the program ran before it stopped.

On BRLESC II, the following error message is always printed when the maximum time is exceeded.

```
EXCEEDED MAX. TIME OR HUNG UP.  NI = ni PJ = pj PPJ = ppj
```

where ni, pj, and ppj are sexadecimal addresses. The ni address is one (sometimes two) more than the address of where the computer stopped or was interrupted, pj is one more than the location of the previous jump instruction and ppj is one more than the address of the jump instruction that was executed before the previous one. This error message line is followed by a line of four consecutive words located at ni - 3 with that address printed at the beginning of the line. If statement number 98765 was used in the main program, BRLESC II jumps to that statement after printing this error message.

Examples:

```
$     MAXT(3)MINS.
$     MAXT(90)SECS.
$     MAXT(2)HRS.
```

Maximum Output

The maximum output specification is of the form:

```
$     MAXO(integer number)LINES
```

where the initial $ is in column 1 and the rest of the specification is in columns 7-72. Blank lines caused by slashes in formats count as lines; however any lines skipped by using vertical space control characters do not count. All tape or card alphanumeric output is included in the counting of lines but binary tape output is not included.

Note that MAXO ends with the letter O, not zero.

When the specified amount of output has been exceeded, BRLESC I/II will not go to statement number 98765.

BRLESC I gives an error print of:

```
EXCEEDED MAXO AT s OCTAL AR.REFS. = a b c CLK = cr
```

where s is the octal address of the statement that caused the output maximum to be exceeded; a,b and c are octal addresses of the last array elements referenced and cr is the clock reading at the time of this error print.

BRLESC II prints the following error message:

```
EXCEEDED MAXO AT e
```

where e is a sexadecimal address that is the location of the last entry to the I/O routine.

Examples:

```
$      MAXO(500)LINES
$      MAXO(20000)LINES
```

It is permissible for MAXO and MAXT specifications to be on the same card with each other or with BRLESC I assembly orders. BRLESC II also allows MAXO and MAXT cards with a minus sign in column 1, but does not allow any assembly orders on the same card.

XXI. STATEMENT NUMBER 98765 AND 98766

The statement number 98765 may be used in a main program on BRLESC I/II to obtain some extra printing after a program fails to run to completion or exceeds the maximum time. When an unexpected machine halt occurs, the operator manually causes BRLESC I/II to go to statement 98765 if this statement number was used in the main program. At 98765, the program should do a limited amount of printing that could be helpful in determining where and why the program stopped and then should execute a STOP statement.

Each link of a CHAIN job on BRLESC I may have its own 98765 statement.

The statement number 98766 may be used in a main program on BRLESC I/II to indicate a statement at which execution should be continued after a "run error" print has occurred. Such error prints are listed in Section XXVI. They are errors detected by predefined functions, subroutines, and the I/O routine. If 98766 is not used in a main program, execution is terminated after a run error print.

Only the last physical link of a CHAIN job on BRLESC I may use 98766 to continue execution after a run error print.

XXII. CHAIN JOBS

BRLESC I allows segmentation of large programs by using CHAIN control cards and a CHAIN subroutine. BRLESC I is essentially compatible with 709/7090 FORTRAN in the way this is done. Each "link" of the chain must be preceded by a control card of

```
*      CHAIN(R,T)
```

where R is an identifying integer number (less than 32768 on 709/7090) and T is a tape unit designation of any two alphanumeric characters (must be B2,B3,A4 or B1 on 709/7090). BRLESC I always uses tape Switch 7 for storing links. Each link consists of a complete FORTRAN program with a main program followed by all of its subprograms. The initial control card should follow the identification card but may precede it. No other control cards are required between links. Only the last link may end with the * DATA card or an E in column 1.

Any link may begin execution of any other link by executing a CALL CHAIN(R,T) statement where R and T both are used to identify the link that is to be executed next.

Data may be passed from one link to the next one through common storage only. No program should assume any other storage is preserved from one link to the next. All links that require correspondence of blank common storage on BRLESC I must have the same total amount of labeled common storage. (BRLESC I assigns blank common storage after all of the labeled common storage has been assigned in the order the labels appeared in COMMON statements.) All links that require correspondence between labeled common blocks must have the label in the same relative position in COMMON statements and the same total length of common storage preceding it. During execution, when one link calls the next link, BRLESC I passes the maximum amount of common storage that was assigned by any link that has previously been executed. In summary, blank common need not be the same length in each link, but each labeled common block does usually have to be the same length.

There is a chance of incompatibility between BRLESC I and other computers if links that have short programs with long common storage are mixed with links that have long programs with short common storage. When this incompatibility arises, an error print of CH.COM.BIF occurs.

Sense switches that are preset with BRLESC I control cards will remain preset in all following links unless the link contains a new preset card. Other control cards will not affect following links except a LIST card will cause dictionary printing in all following links.

DATA statements may not be used in chain jobs on BRLESC I.

XXIII. COMPILING FROM TAPE 12

The BRLESC I/II compilers allow part of a program to be on tape and part of it to be on cards. The card portion may contain changes, deletions and additions to the portion on tape. The card deck also contains control cards that specify when the compiler should switch between compiling from cards and compiling from tape 12.

All of the changes, deletions and additions are accomplished by switching between the card deck and the tape and by skipping lines on the tape.

Three control cards may be used in the card deck to control the compilation from tape 12. (This is tape switch 12 which is the same as FORTRAN unit 9.) The control cards are:

```
                                              Cols. 73-80
*    COMPILE TAPE TO                             ident.
```

This control card causes the compiler to begin compiling from tape 12 and to continue compiling from tape 12 until a line is read that has columns 73-80 identical to columns 73-80 on this control card. This line on tape is not compiled and will not be used when the card deck has another control card that causes compiling from tape to be resumed, i.e., the line is automatically skipped.

```
                                              Cols. 73-80
*    COMPILE TAPE THRU                           ident.
```

This control card causes the compiler to begin compiling from tape 12 and to continue compiling from tape 12 until a line has been compiled that has columns 73-80 identical to columns 73-80 on this control card. If the same program resumes compilation from tape 12, the very next line will be used and no lines are automatically skipped.

```
                                              Cols. 73-80
*    SKIP TAPE THRU                              ident.
```

This control card causes lines to be skipped on tape 12 until a line has been skipped that has columns 73-80 identical to columns 73-80 on this control card. After these lines have been skipped, compilation resumes with the card deck and one of the above control cards must be used if compilation from tape 12 is desired.

It is permissible to use successive skips; the skipping does not stop for any reason other than identity of columns 73-80 and reading the special END TAPE block at the end of the information on a tape.

These three control cards are ignored if they are on tape 12 and should not be put on the tape.

Note that all of the control depends only upon the identification columns of a line. Thus, statements have no special significance and a line include all the statements or maybe just a part of a statement that is on that line. For example, a statement begun on tape could be continued on cards; however a statement on cards cannot be continued on tape because the control card that switches to tape 12 compilation will stop the continuation sequence. Note also that the card deck does not actually rewrite any lines on the tape, it only causes the compiler to use cards instead of tape or to skip lines on the tape.

Data may also be included on the program tape; however, it cannot be changed by control cards before it is read. The program can read it by referring to unit 9 in READ statements. If data is to be read, the last program line on the tape must have been compiled or skipped. Tape 12 is rewound at the end of execution of any program that reads data from it. It is not rewound if it is used only for compilation.

BRLESC I/II will accept tapes that do not have any extra control characters on them, i.e., a simple recording of 80 characters per line with not more than 2000 characters per block. (The block length can vary on the same tape.) BRLESC II will also accept the normal output of a FORAST program but not the output of a FORTRAN program.

XXIV. A PROGRAM TO WRITE PROGRAMS ON TAPE 12

A special program is available for writing FORTRAN programs on a tape (switch 12) so that they can be compiled from tape by BRLESC I/II. This program accepts one or more FORTRAN programs, including data if desired, as input on cards and writes them on tape with a new card number sequence and identification in Columns 73-80.

There are two special control cards that are to be used; one to mark the end of a program and one to mark the end of all the programs. They must be punched with the initial E in column 1 and no blanks other than one blank after END. These two cards are:

END PROGRAM	This card is to be inserted at the end of each complete program deck except it is not required when the END TAPE card is used. It goes after the data when data is included. It causes a file mark to be written on tape 12.
END TAPE	This card is to be inserted at the end of the last program deck that is to be put on tape 12. It goes after data when data is included. It causes the normal end-tape file mark and block to be written and tape 12 to be rewound. If columns 11-16 contain "C.PROB", the computer will immediately begin to compile the next problem which could be a card deck that causes compilation of a program that was just written on tape 12.

This program puts new identification in columns 73-80 of each program line. This consists of 1-6 characters, usually the name of the subprogram followed by an absolute line count that starts at one at the beginning of each subprogram. One or more characters are removed from the end of the name of the subprogram whenever those columns are needed for the line count.

The name of the main program will be "MAIN" if columns 73-78 of the first card are blank. If columns 73-78 of the first card are not blank, they will be used as the name of the main program. The statements are scanned from the beginning of each line so as to obtain the name of a function or subroutine when such a statement is the first statement on a line. The FUNCTION statement may have a type declaration in front of it.

If data is included, the normal * DATA card (or E in col. 1) should be included between the program cards and the data cards. All 80 columns of data cards are copied onto the tape.

If you wish to keep the identification columns as they are on the cards, an * DATA card can be placed in front of the program and then all 80 columns of the program cards will be copied onto the tape. However, that * DATA card must be skipped when compiling the program from the tape and if data appears after the program, an * DATA card must precede the data and must have "DATA" in columns 7-10.

This program is written in FORAST and runs only on BRLESC II. This program will not stop when it reads a PROB card; the END TAPE card must be used to stop it.

XXV. BRLESC I/II COMPILER ERROR PRINTS

The BRLESC I/II compilers check for a limited number of types of errors in programs being compiled. All possible errors are not detected, but some errors will cause one of the error prints listed below. The type of error can be recognized either by the number that follows the word ERROR and precedes the comma or by the "error word" that is printed. The form of the error print is:

FORTRAN ERROR t,m <u>Error Word</u> AT <u>w</u> ON CARD <u>cc</u>

where

t	= type of error; is integer number.
m	= ten col. field at which error was detected; m=0,1, ...,7
Error word	= ten alphanumeric characters that describe the type of error as listed below.
w	= rest of the mth field on the card at time of error detection.

cc = decimal card count of cards read by compiler.

After this error line is printed, another line is printed which is usually the card on which the error occurred, although some undetected errors may later cause an error print at a point where no error occurs. On BRLESC I, when w=m=0, the error was probably on the previous card, rather than on the card that is printed. It should be noted that the probable reason for the error listed below may not be correct, the true error may be quite different.

If any one of the following errors are detected by the compiler, the program will not be executed but a dictionary of names that appeared only once in the program will always be printed. If the LIST(START) control card is being used, the error print will appear between the source statements at the point at which the error was detected; otherwise it will appear before the dictionary.

After an error, compilation is continued until the end of the program is encountered or until 128 errors have been detected.

TYPE	ERROR WORD	DESCRIPTION
1	ILL.CHAR.	Illegal character on program card.
2	SYM.ST.NO.	Symbolic statement number, not all decimal digits.
3	MIXED EXPR	Mixed expression, two operands are not same type.
4	INT**FLT	Integer raised to fl.pt. power is illegal.
5	IL.RETURN	Illegal RETURN statement, used in main program.
6	NO = IN DO	No equals symbol at proper place in DO statement.
7	SUBPRS.>60	Tried to compile more than 60 subprograms. (255 on BRLESC II)
8	BIG ADD.ID	Big address is indexed. Program is too big. (BRLESC I only)
9	NO, CP.GOTO	No comma at proper place in computed GOTO statement.
10	ILL.STAT.	Illegal statement.
11	FLT.INDEX	Subscript involves a fl.pt. number.
12	ILL.DIM.	Number of subscripts is not same as dimensionality of the array.

TYPE	ERROR WORD	DESCRIPTION
13	ILL.COMMA	Comma is used improperly in an arithmetic expression.
14	ASD.ST.NO	Assigned statement number; same statement number used twice, but not in succession.
15	COMPLEX AR	Complex type-statement or I in Column 1.
16	EQU. TABLE	EQUIVALENCE table is full. (BRLESC I allows about 700 different names for whole program. BRLESC II allows about 400 names for each subprogram.)
17	COM. ASGND	COMMON name was previously assigned.
18	ARRAY.REF	Array name used before it was declared.
19	DICT.FULL	Dictionary is full. (about 4000 names.)
20	COL.7 NO.	Statement begins with a decimal digit.
21	SENSE > 6	Sense light or sense switch number greater than 6.
22	DO NO END	Statement number used in DO Statement never appeared. (It may have been missed due to another error.)
24	IL.EQUALS	Illegal = symbol or arithmetic was specified on the left of the = symbol.
25	IL. - BOOL	Illegal "not" operation on boolean card. (BRLESC I only)
26	IL. / BOOL	Boolean division is undefined. (BRLESC I only)
28	IL.**BOOL	Boolean exponentiation is undefined. (BRLESC I only)
29	DRUM STAT.	Drum statements not allowed on BRLESC I/II.
30	IL.IO LIST	Illegal input/output list; usually a name has not been declared an array.
31	FAP CODE	An * FAP card is not allowed on BRLESC I/II.
32	BAD TAPE 7	Temporary tape 7 gives persistent parity errors.
33	NO IDENT *	No identification card at beginning of program.
34	N>10 in NH	Alphanumeric constant of more than ten characters.
35	CONST POOL	The constant pool is full. (1696 arrays and different constants on BRLESC I and 1472 different constants on BRLESC II.)

TYPE	ERROR WORD	DESCRIPTION
36	LABEL COMM	More than 63 different COMMON labels.
37	COM. TABLE	The COMMON table is full. (750 names and labels on BRLESC I, 1000 on BRLESC II within one subprogram or main program.)
38	STP FULL	More than 800 dummy argument references.
39	DOT FULL	More than 63 nested DO loops.
40	ATP FULL	More than 64 dummy argument references in a statement function.
41	ARG FULL	More than 100 subprogram arguments.
42	FTB FULL	More than 50 subroutine names on F cards. (BRLESC I)
43	I>32 in AE	Arithmetic expression has too many operations grouped to the right.
44	ILL.EQUIV	Illegal EQUIVALENCE statement.
45	NON-SEXA.	Illegal character in a sexadecimal word or address.
46	IL.AS.O.T.	Illegal BRLESC I/II assembly order type.
47	IL.AS.ADD	Illegal BRLESC I/II assembly address.
48	NO $ AS.O.	No $ symbol at end of BRLESC I/II assembly order.
49	NON-DEC	Improper character in a decimal number.
50	IL.AS SYN	Illegal SYN statement.
51	DM VAR ID.	Dummy argument was indexed in assembly order. (BRLESC I)
52	NOT , OR)	Improper punctuation.
53	STPE FULL	More than 600 ENTRY dummy argument references.
54	SEL FULL	More than 100 ENTRY names and dummy arguments.
55	DIM. COMMA	Missing comma in DIMENSION.
56	LONG I NO.	Integer constant of more than 17 digits.
57	DUPL. COM.	Duplicated name in COMMON.
58	MAXTO NO I	Number on MAXT or MAXO card is not an integer.
59	EXTRA PUNC	Extra punctuation symbol.
60	BAD L.NAME	Bad logical operation or relation name or illegal period.

TYPE	ERROR WORD	DESCRIPTION
61	DATA BAD =	Equal symbol at illegal place in DATA statement.
62	DAT.IL.NO.	Illegal number in a DATA statement.
63	DATA FULL	Too much data in DATA statements. (About 1000 names and numbers on BRLESC I, 2000 on BRLESC II.)
64	HUNG UP	Computer stopped during compilation.
65	VAR.DIMENS	Variable dimension in EQUIVALENCE.
66	DUMMY VAR.	Dummy argument in EQUIVALENCE.
67	DUM IN COM	Dummy argument in COMMON.
68	CHANGE + -	Obsolete "CHANGE + AND -" control card.
69	========> 24	More than 24 "=" symbols in one statement.
70	AR. NO (,)	Array name not followed by "(" or "," or ")".
71	ERRORS> 128	More than 128 compile errors.
72	AR.FORMAT (	Subscript on array format name.
73	SUBS-NOS< 0	Constant pool and subroutines overlap storage.
74	= IN IF	"=" instead of .EQ. in IF statement

BRLESC II has the following additional error prints:

75	NO $ E.ST	A statement doesn't end where it should.
76	DIM PUNCT.	Improper punctuation in a DIMENSION statement.
77	DUPL.DIM.	A name is declared as an array more than once.
78	SPEC.LATE	A specification statement appeared after the first executable statement.
79	ADJ.DIMENS	Illegal adjustable dimensions; either the array or the dimensions are not dummy arguments.
80	DIMENS > 3	More than three dimensions either declared or used.
81	LABEL > 255	A common label name appears more than 255 times in one subprogram.
82	DICT2.FULL	The second part of the dictionary is full.
83	ST.NO.PT.	A statement number contains a decimal point.
84	COMM.BACK	EQUIVALENCE attempts to extend the beginning of common backward. One of the names from the erroneous equivalence will replace w in the error print.

TYPE	ERROR WORD	DESCRIPTION
85	IL.EQUIV.	Illegal EQUIVALENCE, probably equivalenced two common variables or caused nonconsecutive array elements. One of the names from the erroneous equivalence will replace w in the error print.
86	LAB.LENGTH	A labeled common block is not the same length as it was in some previous subprogram. The label will replace w in the error print.
87	EQUIV.SUB	An EQUIVALENCE subscript has an improper number of dimensions. It must be one or the number declared. The name involved will replace w in the error print.
88	DUM ARG > 60	More than 60 dummy arguments.
89	NOT (OR $	Improper punctuation, should be (or $.
90	NOT , OR $	Improper punctuation, should be , or $.
91	EXT.STAT.	Improper punctuation or name in an EXTERNAL statement.
92	TYPE STAT.	Improper punctuation or name in a type statement.
93	NOT COMMA	Improper punctuation, should be comma.
94	NOT INTEG.	Improper type of name or constant, should be integer.
95	FUN NO DUM	No dummy arguments for function subprogram.
96	LONG STAT.	Too many operations in one statement, statement doesn't end where it should, or more than 199 subscripts in one statement.
97	NO OPERAND	An operand has been omitted.
98	NOT LOGIC=	Name on left of = symbol is logical type when expression on right is not logical.
99	MACH ERROR	Probably a machine error.
100	INT**NO.< 0	Integer quantity raised to a constant negative power.
101	NOT)	Improper punctuation, should be ")".

TYPE	ERROR WORD	DESCRIPTION
102	SHORT STAT	Short statement, appears to be wrong kind of statement, or maybe a specification statement that appears after the first executable or first DATA statement.
103	AR.NOT DIM	Array not dimensioned or statement function is after first executable statement.
104	S.F.DUM > 30	Statement function has more than 30 dummy arguments.
105	NO = IN SF	No equal symbol in what appears to be a statement function.
106	COLS. 1-5	Illegal character in cols. 1-5.
107	END TAPE12	Tried to compile past the end of all information when compiling from tape 12.
108	AS.INC DUM	Had increment on dummy argument in BRLESC II assembly order.

ERROR. DATA STAT. NOT ONE TO ONE CORRESPONDENCE BETWEEN NAMES AND NUMBERS. (Self explanatory.) Both BRLESC I/II also print two additional sexadecimal lines, one that contains some absolute addresses assigned to the names in the DATA statement and a second line that contains the next four numbers (in sexadecimal) that were in DATA statements after the point at which the error was detected.

ERROR. DATA STAT. CANNOT STORE IN LAST 4000 WORDS OF MEMORY Neither computer can store values from DATA statements in the last 4000 words of memory, but only BRLESC II prints an error print when this occurs. BRLESC II also prints the same two lines after this error print as after the "no correspondence" DATA statement error as explained above.

FORTRAN ERROR, INDEXED LARGE ADDRESS. 3 PREV. ORDS. + PART ORDER ON NEXT LINE. A BRLESC I assembly order address larger than 16383 was also indexed.

TAPE 7 REACHED END OF TAPE DURING CHAIN COMPILATION. PLEASE TRY AGAIN.
(Self explanatory, BRLESC I only.)

ERROR TAPE 7 FORTRAN (BRLESC I only)

If the right end of this error print line says "PARITY ERROR", it was caused by a persistent error on temporary tape 7. If the line ends with anything else, it is a name that cannot be found in the dictionary and this usually indicates a machine error.

XXVI. BRLESC I/II RUN ERROR PRINTS

Some of the predefined FORTRAN subroutines and functions used on BRLESC I/II detect certain errors in the data they process. When such an error is detected, a RUN ERROR line is printed and the program is not allowed to continue to run. The error print consists of one line of information of the following form:

RUN ERROR "Error word" Date Cols.1-30 of Ident.Card LE No.

where "Error word" is an alphabetic word that identifies the type of error;

Date	is the date.
LE	is the location of the entry to the subroutine or function that detected the error. It is a decimal address on BRLESC I and a sexadecimal address on BRLESC II.
No.	is a number that in some cases was an illegal argument.

Run Error List: (X and Y represent arguments.)

ERROR WORD	SUBROUTINE	REASON	NO.
LOG X NEG	ALOG	$X \le 0$	X
EXP BIG X	EXP	$X > 354.89$	$X/\log_e 2$
ARCSIN 1+	ARCSIN or ARCCOS	$X > 1+2^{-49}$	$\|X\|$
SINCOS NS	SIN or COS or SINCOS	$\|X\|/2\pi \ge 16^{13}$	$X/2\pi$
POWER OTO-	POWERS (Exponentiation)	$X = 0$ and $Y \le 0$	Zero
CVFTOI BIG	INT or IFIX	$\|X\| \ge 16^{14}$	X
POWER A < 0	POWERS	For X**Y, $X < 0$ when $\|X\| \ge 16.$ or not a multiple of .5	X
LOG10 < = 0	ALOG10	$X \le 0$	X

ERROR WORD	EXPLANATION
END TAPE t	Tried to read beyond binary information written on Fortran tape t.
BAD FORMAT	Illegal character in a FORMAT statement. (Cols. 21-30 of the identification line get replaced with the first ten characters of the bad format.)
NO(FORMAT	More right parentheses than left parentheses in a FORMAT statement.
EXCEEDED MAXT	Exceeded maximum execution time or did not run to normal completion. (See Section XX)
EXCEEDED MAXO	Exceeded maximum amount of output. (See Section XX)
RD.TAPE 8	Attempted to read on tape switch 8, which is the standard output tape and normally is FORTRAN unit 6. The No. at the end of the error line is the FORTRAN unit number.
WR.TAPE 6	Attempted to write on tape switch 6, which is the standard "card" input and normally is FORTRAN unit 5. The No. at the end of the error line is the FORTRAN unit number.
END FILE 6	Attempted to execute an END FILE statement using tape switch 6. The No. at the end of the error line is the FORTRAN unit number.
SKIPFILE 6	Attempted to execute SKIPFILE subroutine using tape switch 6.
SKIPFILE W	Attempted to use SKIPFILE subroutine to move a tape that was last used for writing.
BACKFILE 6	Attempted to execute BACKFILE subroutine using tape switch 6.
BACKFILE 8	Attempted to execute BACKFILE subroutine using tape switch 8.

The following error prints are only on BRLESC II:

ERROR WORD	EXPLANATION
I/O 0H	H type in format preceded by zero or no integer.
I/O EXPBIG	Real input number is too large. (10^{155})
L.LINEREAD	Input or decode line is too long. (160 characters.)
L.LINE PR	Output or encode line is too long. (160 characters, but can't print more than 132 characters.)
I/O SEXANO	Illegal character in sexadecimal input field.

For the above five I/O error prints on BRLESC II, the last number on the error line is the decimal address of the format being used. Also, the ten characters, which normally are columns 21-30 of the identification line, in the error line have been replaced with the first ten characters of the format.

The following error prints are only on BRLESC I:

ERROR WORD	EXPLANATION
LONG LINE	Input or output line is more than 170 characters.
TAPE TKA u	Persistent tape error on trunk A where u is tape switch number and "no." is total number of tape errors.
TAPE TKB u	Same as TAPE TKA u except error is on tape trunk B.
MACH TAPE-	Machine tape error of setting negative sign bits when there wasn't any parity error.
CH.COM.BIG	Illegal combination of large and small COMMON in CHAIN links.
CHAIN ID.	A CHAIN link called is not on the tape.
CHAIN PAR.	Persistent tape 7 parity error when reading a CHAIN link.

XXVII. OPERATION OF BRLESC I/II COMPILERS

The BRLESC I/II compilers each exist on magnetic tape. Many copies of the compiler and the predefined subroutines and functions are on one tape and the tape reading is arranged so that it is checked and automatically corrected by using the next copy on the tape. The tape automatically backs up twenty copies after the last copy on the tape is used. Normally, successive copies are used for compiling successive programs.

Much of the translation is done concurrently with the reading of the program cards (or tape). The partially translated code is kept in memory as much as possible and put on temporary tape switch 7 when the memory is full. The dictionary and constant pool are kept in the memory. After the last card of the program is read (E in Column 1 or * DATA), all unassigned symbols in the dictionary are assigned storage. The memory that will be used by a program is cleared to zeros and then the translation of each instruction is completed and it is stored in the memory for execution. Programs are stored from 01040 on BRLESC I and from 01400 on BRLESC II and may extend to the end of the memory. Next, the subroutines are read from the compiler tape and the ones needed are stored backwards from 0840 on BRLESC I and from 09K0 on BRLESC II. (The standard input/output routines occupy 0860-103L on BRLESC I and 09K0-013LL on BRLESC II.

Neither compiler performs any optimization that requires analysis of the executable flow between statements. They do perform some simple optimizations within statements. These primarily are; performing operations involving only constants at compile time, adding instead of multiplying by two, multiplying by the reciprocal of a constant instead of dividing, using multiplication for raising numbers to small integer powers, and generally avoiding unnecessary and redundant computer instructions. However, the recomputation of common subexpressions is not eliminated and statements are generally performed exactly as written. It is the programmer's responsibility to not write programs that contain gross inefficiencies.

XXVIII. SPEED OF BRLESC I/II FORTRAN COMPILING

The BRLESC I/II compilers are very fast and are designed for "load and go" operation. Programmers are encouraged to keep their FORTRAN programs in symbolic form and translate them each time they are run. This wastes very little if any computer time and is most convenient for the programmer.

Most of the translation is done concurrently with reading the program cards at the present maximum speed of 800 (1500 on BRLESC II) cards per minute. The total time required on BRLESC I for translating a program consisting of C cards can be approximated by the formula:

$$\text{time in secs.} = 2 + C/13 + C/75$$

The 2 seconds is compiler tape time, the C/13 is card read time and C/75 allows time for reading the temporary tape and completing the translation. If the program to be translated is on tape, the C/13 term can be reduced by at least one-half. So the translation rate is about 700 statements per minute from cards and about 2500 statements per minute from tape. (The tape rate will vary from about 1200 to 4000 statements per minute, depending on the complexity and length of the statements being translated.) BRLESC II will compile approximately twice as fast as BRLESC I if printing is not done on-line.

XXIX. CHECKLIST FOR CONVERTING OTHER COMPUTER FORTRAN PROGRAMS TO BRLESC I/II FORTRAN

1. The first card should be an identification card with an asterisk in Column 1. Columns 2-20 should contain a valid BRLESC I/II problem number.

2. Insert MAXO and MAXT cards ($ in column 1) if program writes more than 1200 lines or runs more than 5 minutes.

3. If the signs punched on the input data do not agree with BRLESC I/II signs, the use of a CALL SETMSI(i) statement is required where i = 1 means y punch is minus, i = 2 means x punch is minus, and i = 3 allows either x or y to indicate minus.

4. For BRLESC I, DIMENSION, COMMON and EQUIVALENCE statements must be arranged in that order whenever any variable name appears in more than one of these statements. Also a variable cannot be made equivalent to itself, either directly or indirectly.

5. If there isn't an * DATA card between the program and the input data, insert such a card. (A card with an E in column 1 may be used instead of the * DATA card on BRLESC I/II.)

6. If the program uses sense switches, it is best to insert control cards to preset them. (*SETSSW i UP or DOWN)

7. If tapes are used, make sure the tape unit numbers used are compatible with BRLESC I/II. (Those over 9 may need to be changed.)

8. If desired, change tape output to card output or vice versa by inserting control cards. Since BRLESC I does not have an on-line printer, PRINT statements cause output on tape switch 8 unless changed by control cards.

9. Arrays cannot have more than three dimensions and each reference to an array element must use the same number of subscripts as declared in the array definition.

10. A nonsubscripted array name cannot be used to represent <u>only</u> the first element of the array.

11. For BRLESC I, an array argument for a subprogram must have the same number of dimensions as the corresponding array dummy argument within the subprogram. Such array arguments must not have any subscripts, i.e. the argument must be just the name of the array. For example, it is illegal to try to use a column of a two dimensional array as an argument for a one dimensional array dummy argument.

12. Alphanumeric constants are restricted to a maximum of ten characters except in FORMAT statements.

13. When indexing information is specified within an I/O list, BRLESC I/II do not allow these index variables to be involved in arithmetic subscript operations except for the addition or subtraction of a constant. Indexing control within I/O lists does also actually use the variable named rather than just an unnamed index register as happens on some computers.

14. DO indexing variables actually get used for counting within the DO loop. Some computers may use an unnamed index register instead.

15. The program needs to be modified if it contains any of the following: (1) DRUM or DISC statements, (2) complex arithmetic, (3) assembly instructions for some other computer, (4) more memory or tape units than available on BRLESC I/II, or (5) any of the nonstandard statements and features that are allowed on some other computer and not allowed on BRLESC I/II.

16. If the beginning of an assignment statement is the same as the beginning of some other FORTRAN statement, BRLESC I/II may erroneously assume that it is the other statement. One of the error prints will usually occur when this happens. It is best not to use statement names, such as IF, DO, READ, etc., as variable names,

especially arrays. Also TYPER, TYPEI, TYPEL, TYPED and TYPEC must not be used as the first five letters of any statement except a TYPE statement on BRLESC I. BRLESC II has this restriction only on the first executable statement in each subprogram.

17. ENTRY statements may have to be moved closer to the beginning of a subprogram if any of its dummy arguments will actually be used in a statement that precedes the ENTRY statement. For CDC FORTRAN programs, a list of dummy arguments may need to be added to the ENTRY statement.

18. Nonstandard subprogram exits are not allowed. (Indicated by an * instead of a name on an argument list and integer numbers after RETURN in RETURN statements.)

19. Arithmetic statement functions are made available to a whole program on BRLESC I rather than just the subprogram in which they appear. This shouldn't cause any difficulty unless the same name has been used for two different arithmetic statement functions.

20. If possible, ask the original programmer if any special characteristics of a particular computer or FORTRAN compiler were assumed when writing the program.

21. If possible, run a test case that has been run on another computer.

In addition to the above general comments, programs that were written for other computers can have a number of other incompatibilities. The following list is a list of some of the features that are allowed on at least one other computer but are not allowed on either BRLESC I or BRLESC II:

1. Complex arithmetic.
2. Mixed type arithmetic expressions.
3. DO loops that will not be done at least once.
4. BUFFER IN, BUFFER OUT, IF(EOF,t), IF(IOCHECK,t) and IF(UNIT,t) statements.

5. Executable DATA statements that have statement numbers.
6. Numbers as COMMON labels.
7. Octal constants with a B at the end.
8. ENTRY statements without dummy arguments. CDC automatically uses the subprogram dummy arguments with each ENTRY statement.
9. NAMELIST statements.
10. Type-statements of nonstandard form that declare unusual data structures and operations.
11. Type-statements and DIMENSION statements that contain initial values.
12. Nonstandard RETURN statements, e.g. RETURN 3 and associated CALL statements.
13. IMPLICIT, PARAMETER, DEFINE, DEFINE FILE and FIND statements.
14. T format specification and free format I/O.
15. END and ERR parameters in READ statements.
16. More than three dimensions.
17. INCLUDE, DELETE, EDIT and ABNORMAL statements.
18. .EOR. logical masking operator.
19. GLOBAL and ACCEPT statements.
20. Improper handling of underflow, overflow, rounding, and sign of zero.

XXX. SUMMARY OF BRLESC I/II FORTRAN STATEMENTS

Notations:

s,s1,s2,.....	are statement numbers (look like integer numbers).
i,i1,i2,.....	are integer variable names.
m,m1,m2,.....	are integer variable names or integer constants.
ae	represents an arithmetic expression.
le	represents a logical expression.
b,c,d,e,f	represent any variable names or constants.
t	represents a tape unit number.
f	represents the statement number or array name of a FORMAT statement.
v,v1,v2	represent variable names.

Specification Statements:

DIMENSION v,v1,v2,...	Declares array names and maximum dimensions of each.
EQUIVALENCE (v,v1,..),(v2,v3,...)	Declares synonymous names.
COMMON v,v1,.../ a/v2,.../ b/v3,...	Declares names common between subprograms. a and b are optional labels.
REAL v,v1,v2,...	Declares real (fl.pt.) variable names.
INTEGER v,v1,v2,...	Declares integer variable names.
LOGICAL v,v1,v2,...	Declares logical variable names.
DOUBLE PRECISION v,v1,...	Same as REAL on BRLESC I/II except for BRLESC II common assigning.

COMPLEX v,v1,v2,...	Not allowed on BRLESC I/II.
TYPE REAL v,v1,v2,...	Same as REAL. (nonstandard)
TYPE INTEGER v,v1,v2,...	Same as INTEGER. (nonstandard)
TYPE LOGICAL v,v1,v2,...	Same as LOGICAL. (nonstandard)
TYPE DOUBLE v,v1,v2,...	Same as DOUBLE PRECISION. (nonstandard)
TYPE COMPLEX v,v1,v2,...	Not allowed on BRLESC I/II. (nonstandard)
EXTERNAL name1,name2,...	Specifies names of functions or subroutines that are used as arguments for other funct- ions or subroutines.

Assignment Statements:

v = ae	Evaluates expression ae and stores result in v.
sf(v,v1,...) = ae	Statement function where sf represents its name and v,v1,... are the dummy arguments.
v = le	Evaluates logical expression le and stores .TRUE. or .FALSE. in v. (v must be a logical variable.) (If the operands for a logical operation in le are arithmetic variables, then the operation is performed on all bits of the word except BRLESC I uses only rightmost 65 bits. This masking opera- tion is nonstandard.)

vn = ... = v1 = v = ae or le	Multiple assignment statement. It is equivalent to the statements: v = ae; v1 = v; ... vn = vn-1. All v's must be arithmetic for an arithmetic expression and all logical for a logical expression. (non-standard)

Control Statements:

GO TO s	Execute statement s next.
ASSIGN s TO i	Put address of s into i.
GO TO i, (s1,s2,...)	Execute next the statement whose number was last assigned to i by an ASSIGN statement.
GO TO (s1,s2,...),i	Execute statement si next.
DO s i = m1,m2,m3	Repeat statements to and including s with i = m1,m1 + m3, m1 + 2m3,... until i > m2.
DO s i = m1,m2	Same as above with m3 = 1.
IF(ae)s1,s2,s3	Execute statement s1 next if ae is negative; s2 next if ae is zero and s3 next if ae is positive.
IF(le)st	where st is any executable statement except logical IF and DO. Statement st is done only if le has value .TRUE.

IF(ae or le)s1,s2	Statement s1 is executed next if the expression is not zero or .TRUE. and statement s2 is executed next if the expression is zero or .FALSE. (Is CDC statement.)
CONTINUE	Dummy statement.
STOP or STOP w	End of execution of program. (w is octal no.)
PAUSE or PAUSE w	Computer halts. (Displays octal no. w.)
CALL name (v,v1,v2,...)	Perform the subroutine specified by "name".

Nonstandard Control Statements:

IF(SENSE SWITCH r)s1,s2	Execute statement s1 next if switch r is down, s2 next if it is up.
SENSE LIGHT r	For r = 0 turn all sense lights off. For r = 1,2,3, or 4, turn light r on.
IF(SENSE LIGHT r)s1,s2	Execute statement s1 or s2 next if sense light r is on or off respectively. Turn light r off if it was on.

IF ACCUMULATOR OVERFLOW s1,s2
IF QUOTIENT OVERFLOW s1,s2
IF DIVIDE CHECK s1,s2
	These are special statements to check certain overflow indicators. Statement s1 or s2 is executed next if indicator is on or off respectively.

Subprogram Statements:

SUBROUTINE name (v,v1,v2,...)	Defines the name and beginning of a subroutine. v,v1,v2,... are the dummy arguments.
FUNCTION name (v,v1,v2,...)	Defines the name and beginning of a function subprogram.
RETURN	Indicates an execution exit of a subprogram.
END	Marks the physical end of a subprogram.
BLOCK DATA	Special subprogram statement to allow DATA statements to store into labeled common blocks.
ENTRY name (v,v1,v2,...)	Define the name and dummy arguments of extra entry points for subprograms. (Nonstandard)

Input/Output Statements:

FORMAT (Special Specifications)	Describes the fields for input/output data.
READ(t,f) list	Read alphanumeric tape.
WRITE(t,f) list	Write alphanumeric tape.
READ(t) list	Read binary tape.
WRITE(t) list	Write binary tape.
END FILE t	Write end-of-file mark on tape.
BACKSPACE t	Move tape back one record.
REWIND t	Rewind tape.

Nonstandard I/O Statements:

READ f, list	Read cards. Same as READ(5,f) statement on BRLESC I/II.
PUNCH f, list	Punch cards. May write on tape switch 8 on BRLESC I/II.
PRINT f, list	Originally meant to print on-line. BRLESC I/II write on tape switch 8 (FORTRAN unit 6).
READ INPUT TAPE t,f, list	Read alphanumeric tape.
INPUT t, f, list	Read alphanumeric tape.
WRITE OUTPUT TAPE t, f, list	Write alphanumeric tape.
OUTPUT t, f, list	Write alphanumeric tape.
READ TAPE t, list	Read binary tape.
WRITE TAPE t, list	Write binary tape.

Nonstandard Internal I/O Conversions:

DECODE(m,f,v) list	Decode alphanumeric information into items on list.
ENCODE(m,f,v) list	Encode list items into alpha numeric characters.

Data Initialization Statements:

DATA v,vl,.../c,cl,.../	Stores initial values for variables. c,cl,... represent constants.
DATA (v=c),(vl=cl),...	This is CDC form of the DATA statement. (Nonstandard)

REFERENCES

1. American National Standards Institute Document X3.9-1966, FORTRAN, March 1966.

2. Campbell, L. and Beck, G. The Instruction Code for the BRL Electronic Scientific Computer (BRLESC), Ballistic Research Laboratories Memorandum Report No. 1379, November 1961.

3. Campbell, L. and Beck, G. The FORAST Programming Language for ORDVAC and BRLESC, Ballistic Research Laboratories Report No. 1273, March 1965.

4. IBM 7090/7094 Programming Systems FORTRAN II Programming, (Form C28-6054-5), 1963.

5. IBM 7090/7094 Programming Systems FORTRAN IV Language, (Form C28-6274-2), 1963.

6. FORTRAN 66, CDC 6600 Programming System, Volume 3, 1964.

APPENDIX A. LIST OF PREDEFINED FUNCTIONS FOR BRLESC I/II

(R indicates real and I indicates integer)

(ai indicates ith argument)

II NAME	ST'D NAME	ARGUMENT	RESULT	Number of ARGUMENTS	DEFINITION
ABSF	ABS	R	R	1	Absolute value.
XABSF	IABS	I	I	1	Absolute value.
INTF	AINT	R	R	1	Truncation to whole number.
XINTF	INT	R	I	1	Convert real no. to integer.
MODF	AMOD	R	R	2	a1 (mod A2).
XMODF	MOD	I	I	2	a1 (mod a2).
MAXOF	AMAXO	I	R	≥ 2	Chooses largest argument.
MAX1F	AMAX1	R	R	≥ 2	Chooses largest argument.
XMAXOF	MAXO	I	I	≥ 2	Chooses largest argument.
XMAX1F	MAX1	R	I	≥ 2	Chooses largest argument.
MINOF	AMINO	I	R	≥ 2	Chooses smallest argument.
MIN1F	AMIN1	R	R	≥ 2	Chooses smallest argument.
XMINOF	MINO	I	I	≥ 2	Chooses smallest argument.
XMIN1F	MIN1	R	I	≥ 2	Chooses smallest argument.
FLOATF	FLOAT	I	R	1	Convert integer to real.
XFIXF	IFIX	R	I	1	Convert real to integer.
SIGNF	SIGN	R	R	2	Sign of a2 * | a1 | .
XSIGNF	ISIGN	I	I	2	Sign of a2 * | a1 | .
DIMF	DIM	R	R	2	a1 - minimum (a1,a2).
XDIMF	IDIM	I	I	2	a1 - minimum (a1,a2).
SQRTF	SQRT	R	R	1	Square root.
SINF	SIN	R	R	1	Sine (argument in radians).
COSF	COS	R	R	1	Cosine (argument in radians).
LOGF	ALOG	R	R	1	Natural logarithm.
EXPF	EXP	R	R	1	Exponential.
ATANF	ATAN	R	R	1	Arctangent (result in radians).

II NAME	ST'D NAME	ARGUMENT	RESULT	Number of ARGUMENTS	DEFINITION
TANHF	TANH	R	R	1	Hyperbolic tangent.
	ALOG10	R	R	1	Base ten logarithm.
	ATAN2	R	R	2	Arctangent of (a1/a2).

NAME	ARGUMENT	RESULT	Number of ARGUMENTS	DEFINITION

Nonstandard functions allowed on BRLESC I/II:

NAME	ARGUMENT	RESULT	Number of ARGUMENTS	DEFINITION
XLOCF or LOC	R or I	I	1	Stores the <u>address</u> of a1.
ATAN1	R	R	2	Same as ATAN2.
ARCSIN	R	R	1	Arcsine.
ARCCOS	R	R	1	Arcosine.
ARCTAN	R	R	1	Arctangent (same as ATAN).
ARCCOT	R	R	1	Arcotangent.
SINH	R	R	1	Hyperbolic sine.
COSH	R	R	1	Hyperbolic cosine.
TAN	R	R	1	Tangent.
COT	R	R	1	Cotangent.
SEC	R	R	1	Secant.
CSC	R	R	1	Cosecant.
FRACT	R	R	1	Fractional part of argument.
SIGN1	R or I	R	1	-1.0,0., or 1.0 for a1 < 0, $= 0$, > 0 respectively.

Standard double precision functions allowed on BRLESC I/II with either real or double precision (D) arguments:

NAME	ARGUMENT	RESULT	Number of ARGUMENTS	DEFINITION
DABS	D	D	1	Absolute value.
IDINT	D	I	1	Convert double to integer.
DMAX1	D	D	≥ 2	Chooses largest argument.
DMIN1	D	D	≥ 2	Chooses smallest argument.
DSIGN	D	D	2	Sign of a2 $* \vert a1 \vert$.
SNGL	D	R	1	Convert double to real.

NAME	ARGUMENT	RESULT	Number of ARGUMENTS	DEFINITION
DBLE	R	D	1	Convert real to double.
DEXP	D	D	1	Exponential.
DLOG	D	D	1	Natural logarithm.
DSQRT	D	D	1	Square root.
DSIN	D	D	1	Sine (argument in radians).
DCOS	D	D	1	Cosine (argument in radians).
DATAN	D	D	1	Arctangent (result in radians).
DATAN2	D	D	2	Arctangent of (a1/a2).
DLOG10	D	D	1	Base ten logarithm.
DMOD	D	D	1	a1 (mod a2).

APPENDIX B. THREE EXAMPLES OF FORTRAN PROGRAMS

(PROGRAM,INPUT DATA, AND OUTPUT ARE LISTED)

```
*CS916, EXAMPLE 1.  MULTIPLY ELEMENTS OF TWO VECTORS A*B
        DIMENSION A(10),B(10),C(10)
     2  FORMAT(5E14.3)
        READ(5,2)A,B
        DO 3 I=1,10
     3  C(I)=A(I)*B(I)
        WRITE(6,4)C
        STOP
     4  FORMAT(16H VECTOR PRODUCTS/(1H  ,5E14.7))
        END
*       DATA
            14.1          60.35       22.8        91.7        374.18
            36.2         193.44       83.61       2.648         9.8
            4.21           8.23       15.9        7.77         88.1
             2.7           3.0        8.1118      19.1         42.44
```

```
 JAN.9,70      BRLESC FORTRAN 2 AND 4
*CS916, EXAMPLE 1.  MULTIPLY ELEMENTS OF TWO VECTORS A*B
VECTOR PRODUCTS
0.5936100E 02 0.4966805E 03 0.3625200E 03 0.7125090E 03 0.3296526E 05
0.9774000E 02 0.5803200E 03 0.6782276E 03 0.5057680E 02 0.4159120E 03
```

```
*CS916, EXAMPLE 2.  FIND SMALLEST NUMBER IN ARRAY F
        DIMENSION F(20)
      2 FORMAT(5E14.5)
        READ(5,2)F
        SMALL=F(1)
        DO 9 J=2,20
        IF(SMALL-F(J))9,9,8
      8 SMALL=F(J)
      9 CONTINUE
        WRITE(6,3)SMALL
        STOP
      3 FORMAT(12H SMALLEST F=,F13.6)
        END
*       DATA
            14.1         60.35       22.8       91.7      374.18
            36.2        193.44      83.61      2.648         9.8
            4.21          8.23       15.9       7.77        88.1
             2.7           3.0     8.1118       19.1       42.44
```

```
 JAN.9,70     BRLESC FORTRAN 2 AND 4
*CS916, EXAMPLE 2.  FIND SMALLEST NUMBER IN ARRAY F

SMALLEST F=     2.648000
```

```
*CS916, EXAMPLE 22 FROM BRL REPORT 1209 CODED IN FORTRAN.
C       USE BISECTION METHOD TO FIND ROOT OF F(X)=X**3-X-1 IN INTERVAL
C       (1,2)
     11 FORMAT(1H ,5X,1HX,10X,4HF(X)/)
     21 FORMAT(1H ,1P2E15.7)
     31 FORMAT(25H CONDITIONS NOT SATISFIED)
        X=1.
        X1=2.
        EPS=.00001
        ASSIGN 1 to K
        WRITE(6,11)
     44 F=X*(X*X-1.)-1.
        WRITE(6,21)X,F
        GOTOK,(1,4,7)
      1 FO=F
        IF(F.LT.O.)GOTO 2
     15 XP=X
        GOTO 3
      2 XN=X
      3 X=X1
        ASSIGN 4 TO K
        GOTO 44
      4 F1=F
        IF(F.LT.O.)GOTO 5
     45 XP=X
        GOTO 6
      5 XN=X
      6 ASSIGN 7 to K
        IF(FO*F1)66,65,65
     65 WRITE(6,31)
     67 STOP
     66 X=(XN+XP)/2.
        GOTO 44
      7  IF(ABSF(F).LT.EPS) GOTO 6
     71  IF(F.LT.O.)GOTO 8
     72  XP=X
         GOTO 66
      8  XN=X
         GOTO 66
         END
*        DATA
 JAN.9,70     BRLESC FORTRAN 2 AND 4
*CS916, EXAMPLE 22 FROM BRL REPORT 1209 CODED IN FORTRAN.
     X            F(X)
 1.0000000E 00 -1.0000000E 00
 2.0000000E 00  5.0000000E 00
 1.5000000E 00  8.7500000E-01
 1.2500000E 00 -2.9687500E-01
 1.3750000E 00  2.2460938E-01
 1.3125000E 00 -5.1513672E-02
 1.3437500E 00  8.2611084E-02
 1.3281250E 00  1.4575958E-02
 1.3203125E 00 -1.8710613E-02
 1.3242187E 00 -2.1279454E-03
 1.3261719E 00  6.2088296E-03
 1.3251953E 00  2.0366507E-03
 1.3247070E 00 -4.6594883E-05
 1.3249512E 00  9.9479097E-04
 1.3248291E 00  4.7403882E-04
 1.3247681E 00  2.1370716E-04
 1.3247375E 00  8.3552438E-05
 1.3247223E 00  1.8477852E-05
 1.3247147E 00 -1.4058747E-05
```

APPENDIX C. SYMBOLIC AND SEXADECIMAL BRLESC I ORDER TYPES.

Key:		
	α, β, γ	primary addresses in three address instruction.
	a, b, c	index addresses respectively for α, β, γ.
	A, B, C	effective addresses respectively.
	x, y, z	contents of A, B, C respectively.

Sexa.	Sym.	Action
2	A	$x + y \rightarrow z$
3	S	$x - y \rightarrow z$
4	M	$x * y \rightarrow z$
5	D	$x/y \rightarrow z$
60	C or C-	Jump to C if $x < y$
62	C+	Jump to C if $x \geq y$
7	SQRT	$\sqrt{\lvert x \rvert} \rightarrow z$
8	SHX	Shift x by B → z (See shift code below.)
9	TP	Transplant. Replace part of z with part of x as specified by 1 bits in y and shifted cyclic (not tags) Left 4* parameter.
K	B	Boolean. (See boolean table below.)
S	CB	Jump to C if boolean result = 0.
S6	CEQ	Jump to C if x = y. Same as CB6.
N	CNB	Jump to C if boolean result $\neq$ 0.
N6	CNEQ	Jump to C if $x \neq y$. Same as CNB6.
J2	PMA	Polynomial Multiply. $x * y + z \rightarrow 0$ (zero register)
L	IT	Interpret. Own address → 1, A → 4, B → 5, C → 6 and jump to 040.
00	HALT	Halt
01	SET or SI	Set Indexes. $\alpha \rightarrow a$, $\beta \rightarrow b$, $\gamma \rightarrow c$
02	INC or II	Increase Indexes. A → a, B → b, C → c
03	LP	Loop. Jump to C if loop not yet done B times.
04	J	Jump to C.
05	JS	Jump to subroutine at C. Own address → 1, A → 2, B → 3.

Sexa.	Sym.	Action
06	J+	Jump to C if $x \geq 0$.
07	J-	Jump to C if $x < 0$.
08	TAPE or CARD	I/O. Transfer B words from A according to γ.
08	ZERO	Clear B words beginning at A if $\gamma = 0807$.
09	SIJ	Set indexes, $\alpha \rightarrow a$, $\beta \rightarrow b$, and jump to C.
0K	IIJ	Increase indexes, $A \rightarrow a$, $B \rightarrow b$, and jump to C.
0S	EA	Effective addresses. $A \rightarrow b$ and $C \rightarrow \beta$ where β must be an index address.
0N	JA	Jump to C if rightmost character of x is same as β.
0J	JC	Jump to C if condition specified by β is on.
0F	NOP	No operation.
0L	RSW	Read 68 console switches into B words at A and jump to C.
10	MMF	Move B memory words from A to C. A and C are increased by 1 for each word.
13	LPI	Loop on index a. Jump to C if loop hasn't been done B times after increasing contents of a by 1.
18	MMB	Move B memory words from A to C. A and C are decreased by 1 for each word.
1N	JNA	Jump to C if rightmost character of $x \neq \beta$.
1J	JNC	Jump to C if condition specified by β is off.
1F	MI or IM	Integer multiply. $x * y \rightarrow z$.
1L	RCL	Read clock into A and jump to C. B must be 1.

Boolean Operations

(For B, CB, and CNB orders.)

Key:

*	And
+	Inclusive or
$\bar{x}$	Complement of x

All words are 68 bits.

Dec. Parameter	Result
0	0
1	$\bar{x} * \bar{y}$
2	$\bar{x} * y$
3	$\bar{x}$
4	$x * \bar{y}$
5	$\bar{y}$
6	$x * \bar{y} + \bar{x} * y$ (Excl. or)
7	$\bar{x} + \bar{y}$
8	$x * y$ (And)
9	$x * y + \bar{x} * \bar{y}$
10	y
11	$\bar{x} + y$
12	x
13	$x + \bar{y}$
14	$x + y$ (Inclusive or)
15	1

Symbolic Parameters

(For A, S, M, D, C, SQRT, SH, PM orders.)

Sym.	Par. Bit
X	1 (0 for F)
A or +	2
V	4
R	8

Shift Code

(B of SHX order)

Sexa. Value	Meaning
0 - 07L	Amount of shift
080	Right shift
0100	Clear right 8 bits before shift
0200	Cyclic shift
0400	Round
0800	Include sign and tags
01000	Logical shift
02000	Double Length

γ of I/O Orders

Let γ = C3C2C1 where each C is one sexadecimal digit.

C1 Sexa.	Meaning
3 or 5	Read characters from tape.
4 or 2	Write characters on tape.
S or J	Read binary tape.
N or K	Write binary tape.

C2 Sexa.	
1-L	Tape unit number.

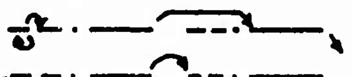
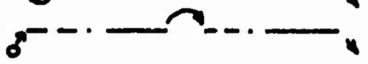
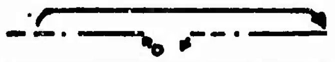
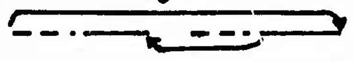
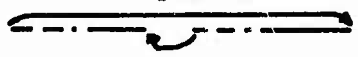
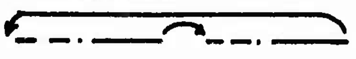
C3 Sexa. for Tape Reading	
0	Read one block.
2	Move forward by blocks.
3	Move backward by blocks.
5	Rewind.
6	Rewind and Unload.
7	Move backward by file marks.
8	Move forward by file marks.

C3 Sexa. for tape writing	
0	Prepare for writing and write a file mark. (Not needed for 1/2" tapes.)
1	Write one block.
2	Write file mark and prepare to read. (Not needed for 1/2" tapes.)
3	Erase backward one block or one file mark. (Not allowed for 1/2" tapes.)
4	Erase forward to leave a blank gap. (Not allowed for 1/2" tapes.)
5	Write a file mark.

See BRL Memorandum Report No. 1379 for additional information on the BRLESC I instruction code.

APPENDIX D. SYMBOLIC AND SEXADECIMAL BRLESC II ORDER TYPES.

Key:	M	Memory Contents	R	R Register Contents
	A	Accumulator Contents	D	D Register Contents
	EA	Effective Address	1	Integer One.
	F	Leading F on Symbolic indicates fl. pt. operation.		
	I	Index Register for indexing the next order.		

Sexa.	Sym.	Action	Sexa.	Sym.	Action
00			32	EA-	A = -EA
02	+1	A = M-1	34	OU'	A = 0 and Jump
04	(-)	A = A-M	36	EA(+)	A = A+EA
06	F(-)	A = A-M	38	LSO	A = 0, then do LSD
08	RS		3K	ANDM	M = A = A $\wedge$ M
OK	RSL		3N	I	I = M
ON	A(-)	A = A-R	3F	IRP	I = EA for Repeat
OF	EA(-)	A = A-EA	40	C'	Jump if A $\geq$ 0
10	M	M = A(65 bits)	42	C'-	Jump if A < 0
12	FM	M = A	44	-HV	A = A-$\lvert$M$\rvert$
14	U'	Jump to right order	46	F-HV	A = A-$\lvert$M$\rvert$
16	MI1	I = A-1	48	LM	M = A(68 bits)
18	LS		4K	A+LM	M = A = R(68 bits)
1K	LSD		4N	A+M	M = A = R(65 bits)
1N	LSC		4F		
1F	I1	I = M-1	50	E'	Set right add.
20	C	Jump if A $\geq$ 0	52	PJE'	Set return add.
22	C-	Jump if A < 0	54	M-A	A = M-A
24	-	A = -M	56	FM-A	A = M-A
26	F-	A = -M	58	-A	A = -A
28	RSC		5K	F-A	A = -A
2K	IORM	M = A = A $\vee$ M	5N	SQRT	A = $\sqrt{A}$
2N	A-	A = -R	5F	FSQRT	A = $\sqrt{A}$
2F			60	CZ	Jump if A = 0
30	OM	M = A = 0	62	CNZ	Jump if A $\neq$ 0

Sexa.	Sym.	Action	Sexa.	Sym.	Action
64	1-1	$A = -\lvert M \rvert$	9F		Set Timer.
66	F 1-1	$A = -\lvert M \rvert$	K0	OU	A = 0 and Jump.
68			K2	EA+	A = EA
6K			K4	+	A = M(65 bits)
6N	A 1-1	$A = -\lvert R \rvert$	K6	F+	A = M
6F			K8	XU	A,R = A*M
70	OE'	A = 0 and do E'.	KK	L+	A = M(68 bits)
72	PPJE'	Set return add.	KN	A+	A = R
74			KF		
76			S0	1M	M = A = 1
78	/	A = A/M	S2	1(+)M	M = A = M+1
7K	F/	A = A/M	S4	R	R = M(65 bits)
7N	/A	A = M/A	S6	FR	R = M
7F	F/A	A = M/A	S8	IX	A = A*M(Integers)
80	CZ'	Jump if A = 0	SK	LR	R = M(68 bits)
82	CNZ'	Jump if $A \neq 0$	SN	U*	Jump to Subroutine.
84	+HV	$A = A + \lvert M \rvert$	SF	EXC	Do orders at M.
86	F+HV	$A = A + \lvert M \rvert$	N0	U	Jump
88	EOR	$A = A \wedge \overline{M} \vee \overline{A} \wedge M$	N2	NOT	$A = \overline{M}$
8K	IOR	$A = A \vee M$	N4	(+)	A = M+A
8N	'A'	A = D	N6	F(+)	A = M+A
8F	RPE'	Save repeat count n-i.	N8		
90	E	Set left add.	NK	ANDN	$A = A \wedge \overline{M}$
92	RPB	Repeat; Adv. all adds.	NN	A(+)	A = A+R
94	RSW	A = Switches.	NF	EAX	A = A*EA(Integers)
96	RPR	Repeat; Adv. right adds.	J0	(-)M	M = A = A-M
98	(+)M	M = A = A+M	J2	F(-)M	M = A = A-M
9K	F(+)M	M = A = A+M	J4		
9N	RCLK	M = Clock.	J6		

Sexa.	Sym.	Action	Sexa.	Sym.	Action
J8	XA	A = A*M	L0	OE	A = 0 and do E.
JK	FXA	A = A*M	L2	IOS	I/O Select
JN	COV	Jump if Overflow.	L4	SIA	I/O Initial Address
JF	COV'	Jump if Overflow.	L6	SFA	I/O Final Add. + 1
F0	ZX	Cond. Halt	L8	SDA	Set Disc Address
F2	NOP	No Operation.	LK		
F4	1 + 1	A = |M|	LN	HALT	Stop
F6	F 1+1	A = |M|	LF		
F8	RER	R = Error bits.			
FK	AND	$A = A \wedge M$			
FN	A 1+1	A = |R|			
FF	SKIP	NOP, I not used.			

Notes:

(1) Jump orders that contain the ' character always jump to the right half of a word.

(2) The following symbolic orders all cause a "secondary transfer of control" when they have a non-zero address:

A(-), A-, -A, F-A, SQRT, FSQRT, A1-1, 'A', A+, A(+), ZX and A1+1.

Such orders cause a jump to the left half of the specified non-zero address after the execution of any orders that follow these orders in the same word. The address of these orders should normally be omitted so that it will be zero.

APPENDIX E. ARDC PRINTER CHARACTERS

DEC. EQUIV.	CARD CODE	BIT CODE	CHAR.	DEC. EQUIV.	CARD CODE	BIT CODE	CHAR.
0	blank	00 0000	blank	32	11	10 0000	-
1	1	00 0001	1	33	11-1	10 0001	J
2	2	00 0010	2	34	11-2	10 0010	K
3	3	00 0011	3	35	11-3	10 0011	L
4	4	00 0100	4	36	11-4	10 0100	M
5	5	00 0101	5	37	11-5	10 0101	N
6	6	00 0110	6	38	11-6	10 0110	O
7	7	00 0111	7	39	11-7	10 0111	P
8	8	00 1000	8	40	11-8	10 1000	Q
9	9	00 1001	9	41	11-9	10 1001	R
10	2-8	00 1010	&	42	11-2-8	10 1010	!
11	3-8	00 1011	=	43	11-3-8	10 1011	$
12	4-8	00 1100	'	44	11-4-8	10 1100	*
13	5-8	00 1101	:	45	11-5-8	10 1101	]
14	6-8	00 1110	>	46	11-6-8	10 1110	;
15	7-8	00 1111	"	47	11-7-8	10 1111	↑
16	12	01 0000	+	48	0	11 0000	0
17	12-1	01 0001	A	49	0-1	11 0001	/
18	12-2	01 0010	B	50	0-2	11 0010	S
19	12-3	01 0011	C	51	0-3	11 0011	T
20	12-4	01 0100	D	52	0-4	11 0100	U
21	12-5	01 0101	E	53	0-5	11 0101	V
22	12-6	01 0110	F	54	0-6	11 0110	W
23	12-7	01 0111	G	55	0-7	11 0111	X
24	12-8	01 1000	H	56	0-8	11 1000	Y
25	12-9	01 1001	I	57	0-9	11 1001	Z
26	12-2-8	01 1010	?	58	0-2-8	11 1010	←
27	12-3-8	01 1011	.	59	0-3-8	11 1011	,
28	12-4-8	01 1100	)	60	0-4-8	11 1100	(
29	12-5-8	01 1101	[	61	0-5-8	11 1101	%
30	12-6-8	01 1110	<	62	0-6-8	11 1110	\
31	12-7-8	01 1111	#	63	0-7-8	11 1111	@

Bit codes 01 1111(#) and 11 1111 (@) are used as end-of-line and ignore characters respectively for variable length lines.

APPENDIX F. LISTING OF FORTRAN SUBPROGRAM CARD DECKS AVAILABLE FROM
             SYSTEMS PROGRAMMING,BLDG.328, RM 213, APG,MD.

```
KEY TO STAT. NO. FIELD,  (I)  INDICATES INPUT ARG., CALLING PROG. SUPPLIES VALUE
                         (R)  INDICATES RESULT. SUBPROGRAM STORES VALUE THERE.
                         (T)  INDICATES TEMPORARY STORAGE.
                         (IR) INDICATES ARGUMENT USED AS INPUT AND RESULT.
                         (F)  INDICATES ARG. USED AS A FUNCTION NAME.
                         (S)  INDICATES ARG. USED AS A SUBROUTINE NAME.
                         (U)  INDICATES ARG. WITH UNUSUAL USAGE.
```

IMPLIED TYPE OF DUMMY ARG. IS REQUIRED TYPE OF ACTUAL ARG., EXCEPT WHERE NOTED.

IMPORTANT BRLESC 1 RESTRICTIONS, (DO NOT APPLY TO BRLESC 2)
   (1) NO. OF DIMENSIONS MUST BE THE SAME BETWEEN ACTUAL AND DUMMY ARGUMENTS.
   (2) DUMMY ARRAY ARG. MUST HAVE ACTUAL ARG. OF JUST ARRAY NAME (NO SUBSCRIPT).

```
      FUNCTION SIGN1(X)                                                 SIGN1  1
C     SIGN1(X)=-1. FOR X.LT.0., 0. FOR X=0., AND +1. FOR X.GT.0.        SIGN1
C        IS ALSO PREDEFINED ON BRLESC 1/2. CARDS ONLY NEEDED ELSEWHERE.

      FUNCTION FACTRL(I)                                                FACTRL 1
C         PRODUCES REAL FACTORIAL OF INTEGER ARGUMENT.                  FACTRL

      FUNCTION SINH(X)                                                  SINH  01
C        IS ALSO PREDEFINED ON BRLESC 1/2. CARDS ONLY NEEDED ELSEWHERE.

      FUNCTION COSH(X)                                                  COSH  01
C        IS ALSO PREDEFINED ON BRLESC 1/2. CARDS ONLY NEEDED ELSEWHERE.

      FUNCTION TAN(X)                                                   TAN   01
C        IS ALSO PREDEFINED ON BRLESC 1/2. CARDS ONLY NEEDED ELSEWHERE.

      FUNCTION COT(X)                                                   COT   01
C        IS ALSO PREDEFINED ON BRLESC 1/2. CARDS ONLY NEEDED ELSEWHERE.

      FUNCTION SEC(X)                                                   SEC   01
C        IS ALSO PREDEFINED ON BRLESC 1/2. CARDS ONLY NEEDED ELSEWHERE.

      FUNCTION CSC(X)                                                   CSC   01
C        IS ALSO PREDEFINED ON BRLESC 1/2. CARDS ONLY NEEDED ELSEWHERE.

      FUNCTION ARCCOT(X)                                                ARCCOT 1
C        IS ALSO PREDEFINED ON BRLESC 1/2. CARDS ONLY NEEDED ELSEWHERE.

          FUNCTION ARCSIN(X)                                            ARCSIN 1
C        IS ALSO PREDEFINED ON BRLESC 1/2. CARDS ONLY NEEDED ELSEWHERE.

      FUNCTION ARCCOS(X)                                                ARCCOS 1
C        IS ALSO PREDEFINED ON BRLESC 1/2. CARDS ONLY NEEDED ELSEWHERE.
```

```
         REAL FUNCTION NDF(X)                                                NDF    1
C     NORMAL DISTRIBUTION FUNCTION.                                          NDF
C     NEED  REAL NDF STATMENT IN CALLING PROGRAM.                            NDF

              FUNCTION  ERF(X)                                               ERF    1
C     NORMAL DISTRIBUTION FUNCTION. SAME AS NDF AND FORAST N.D.F.            ERF

      FUNCTION FINVND(X)                                                     FINVND 1
C INVERSE OF THE NORMAL DISTRIBUTION FUNCTION , NDF.                         FINVND
C HANDBOOK OF MATH FUNCTIONS BUREAU OF STANDARDS PG.933 26.2.23              FINVND

         FUNCTION  URAN31(I)                                                 URAN31 1
          GENERATES UNIFORM RANDOM NUMBER , 0. TO 1.                         URAN31
C (U) I MUST BE INTEGER VARIABLE, NOT A CONSTANT.                            URAN31
C       MUST USE I=0 INITIALLY. MAY USE I=0 TO RESTART SEQUENCE.             URAN31
C       OTHERWISE I SHOULD NOT BE CHANGED OUTSIDE OF URAN31.                 URAN31
C      URAN31 WILL WORK ON ANY COMPUTER THAT USES AT LEAST 31 BIT            URAN31
C        INTEGER ARITHMETIC FOR POSITIVE INTEGERS. IT WILL                   URAN31
C        PRODUCE THE SAME SEQUENCE OF NOS. ON ALL SUCH COMPUTERS.            URAN31

         SUBROUTINE NRAN31(X1,X2,I)                                          NRAN31 1
          GENERATES NORMAL RANDOM NUMBERS (TWO PER ENTRY) WITH A             NRAN31
            MEAN OF ZERO AND A STANDARD DEVIATION OF ONE.                    NRAN31
C (R) X1  IS RANDOM NUMBER.                                                  NRAN31
C (R) X2  IS ANTOHER RANDOM NUMBER.                                          NRAN31
C (U) I  MUST BE INTEGER VARIABLE. IS ONLY USED AS ARGUMENT FOR URAN31.      NRAN31

      SUBROUTINE RKG(DELTA,N,DERIV,Y,DY,Q,J)                                 RKG    1
C           RUNGE-KUTTA-GILL METHOD. SOLVES SYSTEM OF ORD. DIFF. EQS.        RKG
C (I) DELTA IS THE STEP SIZE.                                                RKG    2
C (I) N     IS THE NUMBER OF VARIABLES (INCLUDING INDEPENDENT VARIABLE).RKG    2
C (S) DERIV IS THE NAME OF A SUBROUTINE THAT COMPUTES THE DERIVATIVES.       RKG    2
C                  TWO ARGS.---SUBROUTINE DERIV(Y,DY)---                     RKG
C(IR) Y     IS A 1 DIM. ARRAY CONTAINING THE VARIABLES (N OF THEM).          RKG    2
C (R) DY    IS A CORRESPONDING 1 DIM. ARRAY OF DERIVATIVES (N OF THEM).      RKG    2
C (T) Q     IS A 1 DIM. ARRAY OF CORRECTION TERMS (N OF THEM).               RKG    2
C (I) J     IF J=1 (INITIAL ENTRY), THE Q ARRAY IS CLEARED                   RKG    2
C             AND RETURN IS EXECUTED IMMEDIATELY AFTER COMPUTING             RKG    2
C               INITIAL DERIVATIVES.                                         RKG    2

      SUBROUTINE DVDINT(X,FX,XT,FT,NP,ND)                                    DVDINT 1
C           DOES DIVIDED DIFFERENCE INTERPOLATION.                           DVDINT
C (I) X   IS ARGUMENT FOR WHICH FUNCTIONAL VALUE IS DESIRED.                 DVDINT
C (R) FX  IS NAME OF THE RESULT.                                             DVDINT
C (I) XT  IS ARRAY OF X VALUES.(1 DIMENSIONAL)                               DVDINT
C (I) FT  IS ARRAY OF FUNCTIONAL VALUES.(1 DIMENSIONAL)                      DVDINT
C (I) NP  IS THE NUMBER OF VALUES IN XT AND FT ARRAYS.                       DVDINT
C (I) ND  IS THE NUMBER OF POINTS TO USE FOR EACH INTERPOLATION.             DVDINT

      SUBROUTINE MATMPY(A,B,C,I,J,K,L,M,N)                                   MATMPY 1
C             MATRIX MULTIPLY.  C(I,K)=A(I,J)*B(J,K)                         MATMPY
C (I) A,B    ARE MATRICES (2 DIM.) TO BE MULTIPLIED.                         MATMPY
C (R) C      IS RESULT, C(I,K)=A(I,J)*B(J,K)                                 MATMPY
C (I) I  IS NO. OF ROWS IN A.                                                MATMPY
C (I) J  IS NO. OF COLS. IN A AND NO. OF ROWS IN B.                          MATMPY
C (I) K  IS NO. OF COLS. IN B.                                               MATMPY
C (I) L  IS DECLARED (MAX.) NO. OF ROWS IN A.                                MATMPY
C (I) M  IS DECLARED (MAX.) NO. OF ROWS IN B.                                MATMPY
C (I) N  IS DECLARED (MAX.) NO. OF ROWS IN C.                                MATMPY
C          REST OF RESULT C IS NOT SET TO ZERO.                              MATMPY
```

```
      SUBROUTINE MATINV(A,N,C,NMAX,K,DET)                               MATINV 1
C                MATRIX INVERSION. A=A**(-1)                            MATINV
C(IR)    A IS THE MATRIX AND IS REPLACED BY ITS INVERSE.               MATINV
C (I)    N IS THE OIMENSION OF THE MATRIX.                              MATINV
C (R)    C IS USEO ONLY WHEN K=1 AS DESCRIBED BELOW.                    MATINV
C (I)    NMAX IS THE MAX. NO. OF ROWS OF A AS DECLARED.                 MATINV
C (I)    K DESCRIBES OPTIONS. K=0 MEANS AN N X N MATRIX.K=1 MEANS       MATINV
C          AN N X N MATRIX AND A SINGLE VECTOR AT C.(THE SOLUTION       MATINV
C          VECTOR REPLACES THE C VECTOR). K>=2 MEANS AN N X (N+K-1)     MATIN'
C          MATRIX. (THE (K-1) VECTORS ARE REPLACED BY THE (K-1)         MATINV
C          SOLUTION VECTORS).                                           MATINV
C (R)    DET IS THE VALUE OF THE MATRIX DETERMINANT.                    MATINV
C          WHEN A IS SINGULAR, AN ERROR PRINT AND RETURN WITH THE       MATINV
C          VALUE OF OET SET TO ZERO IS EXECUTED.                        MATINV

      SUBROUTINE MATMPT(A,B,C,I,J,K,MRA,MRB,MRC,NA,NB,NC)               MATMPT 1
C                MULTIPLICATION OF MATRICES OR THEIR TRANSPOSES.        MATMPT
C (I) A,B    ARE MATRICES (2 DIM.) TO BE MULTIPLIED.                    MATMPT
C (R) C      IS RESULT, C(I,K)=A(I,J)*B(J,K)                            MATMPT
C (I) I,J,K  ARE THE DIMENSIONS OF THE MATRICES AS THEY ARE TO BE USED. MATMPT 2
C          (J IS ALWAYS THE COMMON DIMENSION).                          MATMPT 2
C (I) MRA,MRB,MRC  ARE THE MAX. NO. OF ROWS OF A,B,C RESPECTIVELY       MATMPT 2
C                AS DECLARED.                                           MATMPT 2
C (I) NA,NB,NC  ARE OPTIONS TO BE APPLIED TO A,B,C RESPECTIVELY.        MATMPT 2
C     NA,NB,OR NC=0 MEANS TO USE THE RESPECTIVE MATRIX AS IS.           MATMPT 2
C     NA,NB, OR NC=1 MEANS TO USE THE TRANSPOSE OF THE RESPECTIVE       MATMPT 2
C          STORED MATRIX.I,J,K ARE THEN THE DIMENSIONS OF THE TRANSPOSE.MATMPT 2
C     NC=4  MEANS TO ACCUMMULATE.( C = A X B + C)                       MATMPT 2
C     NC=5  MEANS C = (A X B)' + C                                      MATMPT 2

      SUBROUTINE  FNEQS(A,N,C,NMAX,W)                                   FNEQS  1
C     FORM NORMAL EQUATIONS  (FULL N X (N+1) MATRIX).                   FNEQS  2
C (R) A  IS THE MATRIX OF NORMAL EQUATICNS BEING FORMED.                FNEQS  2
C          A MUST BE CLEARED TO ZEROS BEFORE FIRST CALL OF FNEQS.       FNEQS
C (I) N  IS THE NO. OF TERMS(EXCLUDING FUNCTIONAL VALUE).               FNEQS  2
C (I) C  IS A VECTOR CONTAINING THE TERMS OF THE EQUATION INCLUOING     FNEQS  2
C          THE FUNCTIONAL VALUE AS THE LAST TERM.                       FNEQS  2
C (I) NMAX  IS THE MAX. NO. OF ROWS OF A AS DECLARED.                   FNEQS  2
C (I) W  IS THE WEIGHT TO BE APPLIED TO THIS EQUATION.                  FNEQS  2

      SUBROUTINE SIMSON(FUN,RESULT,A,B,EPS,I)                           SIMSON 1
C          USES SIMPSONS METHOD TO COMPUTE INTEGRAL OF A FUNCTION.      SIMSON
C (F) FUN IS AN EXTERNAL FUNCTION THAT COMPUTES THE FUNCTIONAL VALUE    SIMSON 2
C          FOR A GIVEN ARGUMENT.(AN EXTERNAL STATEMENT IS               SIMSON 2
C          NECESSARY IN THE CALLING PROGRAM.)                           SIMSON 2
C (R) RESULT IS THE RESULTING VALUE OF THE INTEGRAL.                    SIMSON 2
C (I) A IS THE LOWER LIMIT OF INTEGRATION.                              SIMSON 2
C (I) B IS THE UPPER LIMIT OF INTEGRATION.                              SIMSON 2
C (I) EPS IS THE RELATIVE ERROR BOUNO.(I.E. 10**(-5) GIVES 5 DIGIT      SIMSON 2
C             ACCURRACY). IF THE VALUE OF I IS ZERO INITIALLY, AN ERROR SIMSON 2
C              PRINT ANO STOP WILL BE EXECUTED WHENEVER EPS IS NOT      SIMSON 2
C            SATISFIEO BY USING THE MAXIMUM NUMBER OF POINTS(8192).     SIMSON 2
C(IR) I      IF I.NE.O INITIALLY, THE VALUE OF I                        SIMSON
C          WILL BE CHANGED BY THE SUBROUTINE ANO SET TO THE FOLLOWING-- SIMSON 2
C     I=1  IF EPS WAS SATISFIED,                                        SIMSON 2
C     I=2  IF EPS WAS NOT SATISFIED BUT BOTH THE VALUE OF THE           SIMSON 2
C         INTEGRAL AND THE ABSOLUTE ERROR < EPS.                        SIMSON 2
C     I=3  IF EPS WAS NOT SATISFIED AND INTEGRAL > EPS > ABSOLUTE ERROR.SIMSON 2
C     I=4  IF EPS WAS NOT SATISFIEO AS A RELATIVE OR ABS. ERROR BOUND.  SIMSON 2
```

```
      SUBROUTINE FNMIN(N,X,FX,FUN,E,EPS,K)                              FNMIN  1
C             FINDS MINIMUM OF A FUNCTION OF MORE THAN ONE VARIABLE.    FNMIN
C (I) N IS THE NUMBER OF VARIABLES. N<11 UNLESS CHANGE DIMENSION STATS. FNMIN  2
C(IR) X IS A LINEAR ARRAY CONTAINING THE INITIAL ESTIMATES OF THE N     FNMIN  2
C        VARIABLES AND AT RETURN CONTAIN THE VALUES AT THE MINIMUM.     FNMIN  2
C (R) FX IS WHERE THE FUNCTIONAL VALUE AT THE MINIMUM WILL BE STORED.   FNMIN  2
C (F) FUN IS THE NAME OF A FUNCTION OF 2 ARGUMENTS--FUN(X,N)--THAT      FNMIN  2
C           COMPUTES THE VALUE OF THE FUNCTION AT X. (AN EXTERNAL       FNMIN  2
C           STATMENT IN THE CALLING PROGRAM IS NECESSARY).              FNMIN  2
C (I) E IS THE NAME OF A SCALAR WHICH IS USED TO DEFINE THE INITIAL     FNMIN  2
C             TRIAL STEP AND THE INITIAL BOUND FOR THE CHANGE IN EACH   FNMIN  2
C             VARIABLE. E>1. (DELX(I)) INITIAL=E*EPS(I) AND             FNMIN  2
C             (DELX(I))MAX.  INITIAL=20*(E*EPS(I)).                     FNMIN  2
C (I) EPS IS A LINEAR ARRAY OF N EPSILONS DEFINING THE ACCURACY         FNMIN  2
C             DESIRED IN EACH OF THE VARIABLES.                         FNMIN  2
C(IR) K   IF K=0  INITIALLY, AN ERROR PRINT AND HALT WILL BE            FNMIN  2
C             EXECUTED WHENEVER CONVERGENCE WITHIN EPS HAS NOT BEEN     FNMIN  2
C             ACHIEVED AFTER 20*N ITERATIONS.IF K IS NOT ZERO INITIALLY, FNMIN  2
C             RETURN IS EXECUTED UPON CONVERGENCE WITH K SET TO 1,      FNMIN  2
C             OR AFTER 20*N ITERATIONS WITH K SET TO 2.                 FNMIN  2

      SUBROUTINE FDMIN(N,X,DX,F,SUB,D,EPS,EPS1,K)                       FDMIN  1
C             FINDS MINIMUM OF A FUNCTION, USES DERIVATIVES.            FDMIN
C               MUST BE FUNCTION OF MORE THAN ONE VARIABLE, I.E. N.GT.1 .
C (I) N IS THE NUMBER OF INDEPENDENT VARIABLES IN THE FUNCTION TO       FDMIN  2
C             BE MINIMIZED.N<11 UNLESS DIMENSION STATEMENTS ARE MODIFIED. FDMIN  2
C(IR) X IS THE LINEAR ARRAY OF VARIABLES.INITIALLY CONTAIN THE          FDMIN  2
C             ESTIMATES OF THE VALUES AT THE MINIMUM.AT RETURN          FDMIN  2
C             CONTAIN THE FINAL VALUES.                                 FDMIN  2
C (T) DX IS A LINEAR ARRAY CONTAINING THE VALUES OF THE N PARTIAL       FDMIN  2
C             DERIVATIVES OF THE FUNCTION EVALUATED AT X BY THE SUB     FDMIN  2
C             PROGRAM.NO INITIAL VALUES REQUIRED.                       FDMIN  2
C (R) F CONTAINS THE VALUE OF THE FUNCTION AT RETURN.                   FDMIN  2
C (S) SUB IS THE NAME OF A SUBROUTINE--SUB(N,X,F,DX)--THAT COMPUTES     FDMIN  2
C             THE FUNCTIONAL VALUE (F) AND DERIVATIVES (DX).            FDMIN  2
C (I) D IS AN ESTIMATE OF THE IMPROVEMENT IN THE VALUE OF THE FUNCTION. FDMIN  2
C             WHEN D=0, ROUTINE ASSUMES THE MIN. VALUE IS NEAR 0.       FDMIN
C (I) EPS IS THE ACCURACY DESIRED IN THE FUNCTION VALUE.                FDMIN  2
C (I) EPS1 IS A CONDITION ON THE INDEPENDENT VARIABLES.                 FDMIN  2
C             ABS(DELTAX(I))/ABS(X(I))<EPS1.IGNORED IF EPS1 VALUE =0.   FDMIN  2
C(IR) K   IF K IS INITIALLY ZERO, AN ERROR PRINT AND STOP WILL BE       FDMIN  2
C             EXECUTED WHEN FUNCTION IS NOT CONVERGING.IF K IS NOT ZERO FDMIN  2
C             INITIALLY,RETURN IS EXECUTED WITH K SET TO 1 WHEN         FDMIN  2
C             CONVERGENCE IS SATISFIED OR K SET TO 2 WHEN THERE IS NOT  FDMIN  2
C             CONVERGENCE.                                              FDMIN  2


      SUBROUTINE BESSEL (X, BF)                                         BESSEL 1
C          COMPUTES FIRST 3 BESSEL FUNCTIONS OF FIRST AND SECOND KIND.  BESSEL
C (I) X   IS ARGUMENT.                                                  BESSEL
C (R) BF  IS LINEAR ARRAY FOR RESULTS---J0,J1,J2,Y0,Y1,Y2.              BESSEL
```

```
      SUBROUTINE RMBGIN  (FX, FI, LL, UL, TOL, PC)                     RMBGIN 1
C              USES ROMBERG METHOD TO COMPUTE INTEGRAL OF A FUNCTION.  RMBGIN
C (F) FX    IS NAME OF FUNCTION SUBPROGRAM--FX(X)--THAT COMPUTES       RMBGIN
C              THE VALUE OF THE FUNCTION AT X.                         RMBGIN
C (R) FI    IS WHERE THE INTEGRAL VALUE WILL BE STORED.                RMBGIN
C (I) LL    IS THE REAL LOWER LIMIT OF INTEGRATION.                    RMBGIN
C (I) UL    IS THE REAL UPPER LIMIT OF INTEGRATION.                    RMBGIN
C (I) TOL   IS THE RELATIVE ERROR TO BE ALLOWED IN THE RESULT,         RMBGIN
C              I.E. 10**(-5) GIVES 5 DIGIT ACCURACY.                   RMBGIN
C (I) PC    IF PC=0, NO WRITING IS DONE AT EACH STEP.                  RMBGIN
C           IF PC.NE.0, RESULTS OF EACH STEP WILL WRITE ON TAPE 6.     RMBGIN
C              WILL GIVE ERROR LINE OUTPUT AND STOP ITERATING          RMBGIN
C              AFTER TEN STEPS. THEN DOES RETURN WITH THAT RESULT.     RMBGIN

      SUBROUTINE GENLSQ(X,NRX,F,M,A,NRA,N,C,R,AF,ERMS,SIG,T,DET,IC)    GENLSQ 1
C               USES FNEQS AND MATINV SUBROUTINES.(MUST INCLUDE CARDS.) GENLSQ
C (I) X    IS A MATRIX OF TERMS OF EQUATIONS.                          GENLSQ
C (I) NRX IS NUMBER OF ROWS IN X.                                      GENLSQ
C (I) F    IS A VECTOR OF FUNCTION VALUES FOR EQUATIONS.               GENLSQ
C (I) M    IS NUMBER OF EQUATIONS.                                     GENLSQ
C (T) A    IS A MATRIX OF AT LEAST (N)X(N+1),IS REPLACED BY INVERSE.   GENLSQ
C (I) NRA IS NUMBER OF ROWS IN A.                                      GENLSQ
C (I) N    IS NUMBER OF TERMS NOT INCLUDING FUNCTION VALUE, N.LE.99 .  GENLSQ
C (R) C    IS A VECTOR FOR N COEFFICIENTS.                             GENLSQ
C (R) R    IS A VECTOR FOR M RESIDUALS.                                GENLSQ
C (R) AF   IS A VECTOR FOR M APPROXIMATE FUNCTIONS.                    GENLSQ
C (R) ERMS IS THE ROOT MEAN SQUARE ERROR,EQUALS ZERO IF M.LE.N.        GENLSQ
C (R) SIG IS A VECTOR FOR N SIGMAS.                                    GENLSQ
C             SIG IS INVERSE ELEMENT IF INV. ELEMENT IS NEGATIVE.      GENLSQ
C (R) T    IS A VECTOR FOR N T VALUES.                                 GENLSQ
C (R) DET IS THE VALUE OF THE DETERMINANT.                             GENLSQ
C (I) IC   IS THE CONTROL--                                            GENLSQ
C        IC IS 0   COMPUTE EVERYTHING.                                 GENLSQ
C        IC IS 1   COMPUTE ONLY COEFFICIENTS.                          GENLSQ
C        IC IS 2  COMPUTE EVERYTHING EXCEPT RESIDUALS AND APPROXIMATIONS.GENLSQ
C        IC IS 3  COMPUTE EVERYTHING EXCEPT APPROXIMATIONS.            GENLSQ

      SUBROUTINE POLYLS(X,F,M,A,NRA,N,C,R,AF,ERMS,SIG,T,DET,IC)        POLYLS 1
C               USES FNEQS AND MATINV SUBROUTINES.(MUST INCLUDE CARDS.) POLYLS
C (I) X    IS A VECTOR OF INDEPENDENT VARIABLE.                        POLYLS
C (I) F    IS A VECTOR OF FUNCTION VALUES FOR POLYNOMIALS.             POLYLS
C (I) M    IS NUMBER OF POLYNOMIALS                                    POLYLS
C (T) A    IS THE MATRIX OF AT LEAST (N+1)X(N+2),IS REPLACED BY INVERSE POLYLS
C (I) NRA IS NUMBER OF ROWS IN A                                       POLYLS
C (I) N    IS DEGREE OF POLYNOMIAL, N.LE. 99                           POLYLS
C (R) C    IS VECTOR FOR (N+1)COEFFICIENTS                             POLYLS
C (R) R    IS VECTOR FOR M RESIDUALS                                   POLYLS
C (R) AF   IS VECTOR FOR M APPROXIMATE FUNCTIONNS                      POLYLS
C (R) ERMS IS THE ROOT MEAN SQUARE ERROR,ZERO IF M.LE.N+1              POLYLS
C (R) SIG IS A VECTOR FOR N+1 SIGMAS.                                  POLYLS
C             SIG IS INVERSE ELEMENT IF INV. ELEMENT IS NEGATIVE.      POLYLS
C (R) T    IS A VECTOR FOR N+1 T VALUES.                               POLYLS
C (R) DET IS THE VALUE OF THE DETERMINANT.                             POLYLS
C (I) IC   IS THE CONTROL--                                            POLYLS
C        IC IS 0   COMPUTE EVERYTHING.                                 POLYLS
C        IC IS 1   COMPUTE ONLY COEFFICIENTS.                          POLYLS
C        IC IS 2   COMPUTE EVERYTHING EXCEPT RESIDUALS AND APPROXIMATIONSPOLYLS
C        IC IS 3   COMPUTE EVERYTHING EXCEPT APPROXIMATIONS.           POLYLS
```

```
      SUBROUTINE KUTMER(DNXT,DPST,DMAX,N,Y,YP,DERIV,ER,ME,W,ITYP,PS,    KUTMER 1
     1                   PV,PRIN,TC,NTC,NTS,TERM)                         KUTMER 2
C             SOLVES SYS. OF ORD. DIFF. EQS.  USES KUTTA-MERSON METHOD.   KUTMER
C(IR) DNXT STEP SIZE TO BE USED FOR NEXT STEP. THE SIGN OF DNXT WILL     KUTMER
C              DETERMINE THE DIRECTION OF INTEGRATION. DNXT MUST .NE. 0.. KUTMER
C (R) DPST THE STEP SIZE USED FOR THE STEP TAKEN LAST.                    KUTMER
C (I) DMAX ABSOLUTE VALUE OF THE MAXIMUM STEP SIZE TO BE USED.            KUTMER
C (I) N    NUMBER OF EQUATIONS TO BE INTEGRATED. N MUST INCLUDE           KUTMER
C              THE INDEPENDENT VARIABLE. MUST HAVE N. LE. 50              KUTMER
C(IR) Y    ONE DIMENSIONAL ARRAY OF LENGTH N WHICH CONTAINS THE           KUTMER
C              INTEGRALS. ONE OF THE ELEMENTS OF Y MUST BE THE IND. VAR.  KUTMER
C (R) YP   ONE DIMENSIONAL ARRAY OF LENGTH N WHERE THE DERIVATIVES        KUTMER
C              ARE STORED BY ROUTINE DERIV. THE ELEMENTS OF YP MUST BE    KUTMER
C              THE DERIVATIVE OF THE CORRESPONDING ELEMENT OF Y.          KUTMER
C (S) DERIV  SUBROUTINE WHICH COMPUTES THE DERIVATIVES YP. THE FIRST      KUTMER
C              STATEMENT MUST BE OF THE FORM SUBROUTINE DERIV(Y,YP).      KUTMER
C              Y AND YP MUST BE DECLARED ONE DIMENSIONAL ARRAYS IN DERIV. KUTMER
C              THE ACTUAL NAME MUST APPEAR IN AN EXTERNAL STATEMENT.      KUTMER
C (I) ER   IF ER = 0. DO NOT ADJUST THE STEP SIZE.                        KUTMER
C              IF ER .NE. 0. ADJUST THE STEP SIZE SO THAT THE MAXIMUM     KUTMER
C              ERROR WILL BE APPROXIMATLY ER.                             KUTMER
C (I) ME   IF ME = 1 USE ABSOLUTE ERROR TO COMPUTE THE NEW STEP SIZE.     KUTMER
C              IF ME = 2 USE ABSOLUTE ERROR TO COMPUTE THE NEW STEP SIZE  KUTMER
C                        IF THE INTEGRALS ARE .LE. 1.. AND USE RELATIVE   KUTMER
C                        ERROR TO COMPUTE THE NEW STEP SIZE IF THE        KUTMER
C                        INTEGRALS ARE .GT. 1..                           KUTMER
C              IF ME = 3 USE THE WEIGHTS STORED IN ARRAY W ALONG WITH THE KUTMER
C                        ABSOLUTE AND RELATIVE ERRORS TO COMPUTE THE NEW  KUTMER
C                        STEP SIZE.                                       KUTMER
C (I) W    ONE DIMENSIONAL ARRAY OF LENGTH 2*N TO USE WHEN ME = 3.        KUTMER
C              W IS NOT USED WHEN ME = 1 OR 2 (BUT MUST BE SPECIFIED).    KUTMER
C (I) ITYP IF ITYP =0 RETURN FROM KUTMER WHEN SOME TC HAS CHANGED SIGN.  KUTMER
C            IF ITYP .GT. 0 THE VARIABLES FOLLOWING ITYP ON THE DUMMY     KUTMER
C              VARIABLE LIST ARE NOT USED (BUT MUST BE SPECIFIED).        KUTMER
C            IF ITYP = 1 RETURN FROM KUTMER AFTER THE INITIAL DERIVATIVESKUTMER
C              HAVE BEEN COMPUTED.                                        KUTMER
C            IF ITYP .GT. 1 RETURN FROM KUTMER AFTER EACH STEP. THE       KUTMER
C              INTEGRALS HAVE BEEN EVALUATED USING THE VALUES OF THE      KUTMER
C              INTEGRALS AT THE END OF THE STEP.                          KUTMER
C (I) PS   PRINT STEP. MUST HAVE PS .GT. 0. (SEE PRIN).                   KUTMER
C (T) PV   PRINT VARIABLE. MUST BE DEFINED IN SUBROUTINE TERM (SEE PRIN)KUTMER
C (S) PRIN PRINT (OR STORE) SUBROUTINE. PRIN WILL BE REFERENCED           KUTMER
C              1) AFTER THE INITIAL DERIVATIVES HAVE BEEN COMPUTED.       KUTMER
C              2) PV HAS CHANGED BY THE AMOUNT PS.                        KUTMER
C              3) SOME TC HAS CHANGED SIGN.                               KUTMER
C              THE FIRST STATEMENT MUST BE OF THE FORM                    KUTMER
C              SUBROUTINE PRIN (Y,YP). Y AND YP MUST BE ARRAYS IN PRIN.   KUTMER
C              THE ACTUAL NAME MUST APPEAR IN AN EXTERNAL STATEMENT.      KUTMER
C (T) TC   ONE DIMENSIONAL ARRAY OF LENGTH NTC. THE ELEMENTS OF TC        KUTMER
C              MUST BE COMPUTED IN SUBROUTINE TERM. KUTMER WILL RETURN    KUTMER
C              WHEN TC(NTS) CHANGES SIGN.                                 KUTMER
C (I) NTC  THE NUMBER OF ELEMENTS OF TC COMPUTED IN SUBROUTINE TERM.      KUTMER
C              NTC MUST BE .LE. 25 OR THE PROGRAM MUST BE CHANGED.        KUTMER
C (R) NTS  THE SUBSCRIPT OF THE TC WHICH CHANGED SIGN AND CAUSED          KUTMER
C              TERMINATION OF INTEGRATION.                                KUTMER
C (S) TERM TERMINATION ROUTINE. TERM IS REFERENCED AT THE END OF EACH     KUTMER
C              INTEGRATION STEP. IT MUST DEFINE PV AND THE FIRST NTC      KUTMER
C              ELEMENTS OF TC. THE FIRST STATEMENT MUST BE OF THE FORM    KUTMER
C              SUBROUTINE TERM(Y,YP,TC,PV). Y, YP AND TC MUST BE ARRAYS.  KUTMER
C              ACTUAL ARG. MUST BE IN EXTERNAL STAT. IN CALLING PROGRAM.  KUTMER
```

```
      FUNCTION GAMMA(Z)                                                 GAMMA   1

      SUBROUTINE POLYR(N,COEFF,ROOTS,D)                                 POLYR   1
C              FINDS ALL ROOTS OF A POLYNOMIAL.                         POLYR
C (I) N        IS THE DEGREE OF THE POLYNOMIAL.                         POLYR
C (I) COEFF IS VECTOR OF N+1 COEFFICIENTS. CONSTANT TERM IN COEFF(N+1). POLYR
C (R) ROOTS IS ARRAY (2-DIM., 2 BY N) OF THE ROOTS. REAL PARTS IN       POLYR
                 FIRST ROW, IMAGINARY PARTS IN SECOND ROW.              POLYR
C (R) D        IS VECTOR OF N+1 RECOMPUTED COEFFICIENTS.CONSTANT IN D(N+1).POLYR

      SUBROUTINE TALLY(A,NOB,NV,MRA,S,TOTAL,AVG,SD,VMIN,VMAX)           TALLY   1
C              COMPUTES THE TOTAL,MEAN,STANDARD DEVIATION,MIN. AND MAX. TALLY
C                FOR EACH VARIABLE IN A SET(SUBSET) OF OBSERVATIONS.    TALLY
C (I) A        IS THE MATRIX OF OBSERVATIONS,NOB ROWS BY NV COLUMNS.    TALLY
C (I) NOB      IS THE NUMBER OF OBSERVATIONS OF EACH VARIABLE.          TALLY
C (I) NV       IS THE NUMBER OF VARIABLES.                              TALLY
C (I) MRA      IS THE MAX. NO. OF ROWS OF A AS DECLARED.                TALLY
C (I) S        IS A VECTOR OF LENGTH NOB SPECIFYING A SUBSET OF A. ONLY TALLY
C                OBSERVATIONS CORRESPONDING TO NONZERO S(J) ARE CONSIDERED.TALLY
C (R) TOTAL IS A VECTOR OF TOTALS OF EACH VARIABLE.                     TALLY
C (R) AVG      IS A VECTOR OF MEANS OF EACH VARIABLE.                   TALLY
C (R) SD       IS A VECTOR OF STANDARD DEVIATIONS OF EACH VARIABLE.     TALLY
C (R) VMIN     IS A VECTOR OF MINIMUM SAMPLE VALUES OF EACH VARIABLE.   TALLY
C (R) VMAX     IS A VECTOR OF MAXIMUM SAMPLE VALUES OF EACH VARIABLE.   TALLY

      SUBROUTINE TTEST(A,NA,B,NB,NOP,NDF,ANS)                           TTEST   1
C            COMPUTES T-STATISTIC FOR HYPOTHESIS OF EQUALITY OF POPULATION TTEST
C              MEANS,GIVEN INFORMATION CONCERNING POPULATION VARIANCE.  TTEST
C (I) A     IS A VECTOR OF LENGTH NA CONTAINING SAMPLE DATA.            TTEST
C (I) NA    IS THE NUMBER OF OBSERVATIONS IN A.                         TTEST
C (I) B     IS A VECTOR OF LENGTH NB CONTAINING SAMPLE DATA.            TTEST
C (I) NB    IS THE NUMBER OF OBSERVATIONS IN B.                         TTEST
C (I) NOP   IS THE HYPOTHESIS OPTION NUMBER.                            TTEST
C              NOP=1. POPULATION MEAN OF B = SPECIFIED VALUE A(USE NA=1). TTEST
C              NOP=2. POPULATION MEAN OF B = POPULATION MEAN OF A,GIVEN TTEST
C                VARIANCE OF B = VARIANCE OF A.                         TTEST
C              NOP=3. POPULATION MEAN OF B = POPULATION MEAN OF A,GIVEN TTEST
C                VARIANCE OF B NOT EQUAL VARIANCE OF A.                 TTEST
C              NOP=4. POPULATION MEAN OF B = POPULATION MEAN OF A,GIVEN NO TTEST
C                INFORMATION CONCERNING VARIANCE(USE NA=NB).            TTEST
C (R) NDF IS THE NUMBER OF DEGREES OF FREEDOM.                          TTEST
C (R) ANS IS THE T-STATISTIC.                                           TTEST
```

```
      SUBROUTINE TAB1(A,NOB,MRA,S,NOVAR,UBO,FREQ,PCT,STATS)            TAB1    1
C           COMPUTES FOR ONE VARIABLE IN AN OBSERVATION MATRIX(OR A     TAB1
C             MATRIX SUBSET) THE FREQUENCY AND PCT. FREQUENCY OVER CLASSTAB1
C             INTERVALS. FINDS THE TOTAL,MEAN,STANDARD DEVIATION,MIN.   TAB1
C             AND MAX. OF THE SAMPLE.                                   TAB1
C (I) A     IS THE MATRIX OF OBSERVATIONS,NOB ROWS,ONE COLUMN PER VAR.  TAB1
C (I) NOB   IS THE NUMBER OF OBSERVATIONS OF THE VARIABLE.              TAB1
C (I) MRA   IS THE MAX. NO. OF ROWS OF A AS DECLARED.                   TAB1
C (I) S     IS A VECTOR OF LENGTH NOB SPECIFYING A SUBSET OF A. ONLY    TAB1
C             OBSERVATIONS CORRESPONDING TO NONZERO S(J) ARE CONSIDERED.TAB1
C (I) NOVAR IS THE VARIABLE TO BE TABULATED,SPECIFIED COLUMN OF A.      TAB1
C (I) UBO   IS A VECTOR CONTAINING THE LOWER LIMIT,NO. OF INTERVALS,AND TAB1
C             THE UPPER LIMIT OF THE VARIABLE TO BE TABULATED IN UBO(1),TAB1
C             UBO(2),UBO(3) RESPECTIVELY. NUMBER OF INTERVALS,UBO(2),   TAB1
C             MUST INCLUDE TWO CELLS FOR VALUES UNDER AND ABOVE LIMITS. TAB1
C             VECTOR LENGTH IS 3. IF LOWER LIM=UPPER LIM,THE PROGRAM    TAB1
C             USES THE MIN. AND MAX. VALUES OF THE SAMPLE.              TAB1
C (R) FREQ  IS A VECTOR OF FREQUENCIES. VECTOR LENGTH IS UBO(2).        TAB1
C (R) PCT   IS A VECTOR OF RELATIVE FREQUENCIES.VECTOR LENGTH IS UBO(2).TAB1
C (R) STATS IS A VECTOR OF SUMMARY STATISTICS. TOTAL,MEAN,STANDARD      TAB1
C             DEVIATION,MIN. AND MAX. OF THE SAMPLE.(5 VALUES)          TAB1

      SUBROUTINE CORRE(A,NOB,NV,MRA,MRC,MRR,AVG,STD,R,C)               CORRE   1
C         COMPUTES MEANS,STANDARD DEVIATIONS,SUMS OF CROSS-PRODUCTS OF  CORRE
C           DEVIATIONS,AND PRODUCT-MOMENT CORRELATION COEFFICIENTS.     CORRE
C (I) A   IS THE MATRIX OF OBSERVATIONS,NOB ROWS BY NV COLUMNS.         CORRE
C (I) NOB IS THE NUMBER OF OBSERVATIONS OF EACH VARIABLE.               CORRE
C (I) NV  IS THE NUMBER OF VARIABLES.                                   CORRE
C (I) MRA IS THE MAXIMUM NUMBER OF ROWS OF A AS DECLARED.               CORRE
C (I) MRC IS THE MAXIMUM NUMBER OF ROWS OF C AS DECLARED.               CORRE
C (I) MRR IS THE MAXIMUM NUMBER OF ROWS OF R AS DECLARED.               CORRE
C (R) AVG IS A VECTOR OF MEANS OF EACH VARIABLE.                        CORRE
C (R) STD IS A VECTOR OF STANDARD DEVIATIONS OF EACH VARIABLE.          CORRE
C (R) R   IS THE NV BY NV MATRIX OF SUMS OF CROSS-PRODUCTS OF           CORRE
C           DEVIATIONS FROM MEANS.                                      CORRE
C (R) C   IS THE NV BY NV MATRIX OF CORRELATION COEFFICIENTS.           CORRE

      FUNCTION EXPRN(P)                                                EXPRN   1
C       GENERATES EXPONENTIALLY DISTRIBUTED RANDOM NUMBERS.             EXPRN
C         USES FUNCTION URAN31(I).                                      EXPRN
C (I) P IS THE PARAMETER OF THE DISTRIBUTION.                           EXPRN

      FUNCTION IBINRN(P,N)                                             IBINRN  1
C       GENERATES BINOMIALLY DISTRIBUTED INTEGER RANDOM NUMBERS.        IBINRN
C         USES FUNCTION URAN31(I).                                      IBINRN
C (I) P IS THE DISTRIBUTION PARAMETER.                                  IBINRN
C (I) N IS THE DISTRIBUTION PARAMETER DENOTING MAXIMUM VALUE OF THE     IBINRN
C         RANDOM VARIABLE.                                              IBINRN

      FUNCTION IPOSRN(FL)                                              IPOSRN  1
C          GENERATES INTEGER RANDOM NUMBERS FOLLOWING A POISSON         IPOSRN
C            DISTRIBUTION. USES FUNCTION URAN31(I).                     IPOSRN
C (I) FL IS THE PARAMETER OF THE DISTRIBUTION.                          IPOSRN
```

```
      SUBROUTINE PLPRG(X,Y,NPTS,SY1,SY2)                                PLPRG
C     PLOTS IN 56 ROWS AND 90 COLUMNS ON A LINE PRINTER.                PLPRG
C     USES PLPRS SUBROUTINE. (MUST INCLUDE CARDS.) USES ENCODE.(10 CH.) PLPRG
C     THE DATA POINTS ARE CONNECTED BY STRAIGHT LINES.                  PLPRG
C     AXES ARE PLOTTED AND THE MAX. AND MIN. VALUES OF                  PLPRG
C        X AND Y ARE PRINTED BELOW THE PLOT.                            PLPRG
C     THE TIC MARK INTERVALS IN X AND Y ARE PRINTED BELOW THE PLOT.     PLPRG
C     THE TITLES STORED AT SY1 AND SY2 ARE PRINTED ABOVE THE PLOT.      PLPRG
C     A SAMPLE OF THE X,Y DATA IS PRINTED TO THE RIGHT OF THE PLOT.     PLPRG
C (I) X    IS LINEAR ARRAY HOLDING THE X VALUES.                        PLPRG
C (I) Y    IS LINEAR ARRAY HOLDING THE Y VALUES.                        PLPRG
C (I) NPTS IS THE NUMBER OF (X,Y) PAIRS OF VALUES.                      PLPRG
C (I) SY1  IS A LINEAR ARRAY HOLDING FIRST TITLE.                       PLPRG
C             THE LAST CHARACTER MUST BE > .                            PLPRG
C (I) SY2  IS A LINEAR ARRAY HOLDING SECOND TITLE.                      PLPRG
C             THE LAST CHARACTER MUST BE > .                            PLPRG

      SUBROUTINE PLPRS(NXB,NYB,XOR,YOR,XS,YS)
C (I) NXB  IS PAGE REFERENCE POINT IN COLUMNS, FROM THE LEFT. 1<NXB<130 PLPRS
C (I) NYB  IS PAGE REFERENCE POINT IN ROWS, FROM THE BOTTOM.  1<NYB<60  PLPRS
C (I) XOR  IS X DATA COORDINATE OF PAGE REFERENCE POINT.                PLPRS
C (I) YOR  IS Y DATA COORDINATE OF PAGE REFERENCE POINT.                PLPRS
C (I) XS   IS X SCALE. DATA UNITS PER COLUMN.                           PLPRG
C (I) YS   IS Y SCALE. DATA UNITS PER ROW.                              PLPRS

      ENTRY PLPRD(MODE,NCHAR,X,Y,NPTS)                                  PLPRD
C (I) MODE=0 MEANS PLOT GIVEN DATA ONLY.                                PLPRD
C     MODE=1 MEANS PLOT GIVEN DATA POINTS CONNECTED BY STRAIGHT LINES.  PLPRD
C (I) NCHAR  IS PRINTER CHARACTER TO BE USED.                           PLPRD
C              RIGHT ADJUSTED ALPHANUMERIC OR INTEGER EQUIVALENT.       PLPRD
C (I) X      IS LINEAR ARRAY CONTAING X VALUES TO PLOT.                 PLPRD
C (I) Y      IS LINEAR ARRAY CONTAING Y VALUES TO PLOT.                 PLPRD
C (I) NPTS   IS NUMBER OF (X,Y) PAIRS OF VALUES.                        PLPRD

      ENTRY PLPRDA(MODE,NCHAR,X,Y,NPTS,XMIN,XMAX,YMIN,YMAX)             PLPRDA
C     SAME AS PLPRD EXCEPT DATA POINTS OUTSIDE THE LIMITS SPECIFIED     PLPRDA
C           BY XMIN,XMAX,YMIN,YMAX ARE NOT PLOTTED.                     PLPRDA

      ENTRY PLPRA(DX,DY,XMIN,XMAX,YMIN,YMAX,NCHAR)                      PLPRA
C     PLOT AXES INTERSECTING AT(NXB,NYB).                               PLPRA
C (I) DX  IS INTERVAL FOR TIC MARKS ALONG X AXIS.                       PLPRA
C (I) DY  IS INTERVAL FOR TIC MARKS ALONG Y AXIS.                       PLPRA
C (I) XMIN,XMAX ARE AXIS LIMITS IN X.                                   PLPRA
C (I) YMIN,YMAX ARE AXIS LIMITS IN Y.                                   PLPRA
C (I) NCHAR  IS PRINTER CHARACTER TO BE USED.                           PLPRA
C              RIGHT ADJUSTED ALPHANUMERIC OR INTEGER EQUIVALENT.       PLPRA

      ENTRY PLPRT(SYM,XP,YP,NDIR)                                       PLPRT
C         PRINTS TITLES.                                                PLPRT
C (I) SYM  IS LINEAR ARRAY HOLDING CHARACTERS TO BE PRINTED.            PLPRT
C              BLANK CHARACTER IS NOT IGNORED. LAST CHARACTER MUST BE >.PLPRT
C (I) XP,YP ARE DATA COORDINATES OF POSITION OF FIRST CHARACTER.        PLPRT
C (I) NDIR=0  MEANS PRINT HORIZONTALLY.                                 PLPRT
C     NDIR=1  MEANS PRINT VERTICALLY.                                   PLPRT

      ENTRY PLPRP                                                       PLPRP
         CAUSES THE ENTIRE PAGE TO BE PRINTED.                          PLPRP
      SUBROUTINE TAB2(A,NOB,MRA,MRB,S,NOVAR,UBO,FREQ,PCT,STAT1,STAT2)   TAB2    1
C             PERFORMS A TWO-WAY CLASSIFICATION FOR TWO VARIABLES IN AN  TAB2
C               OBSERVATION MATRIX(OR A MATRIX SUBSET) OF THE FREQUENCY, TAB2
C               PCT. FREQUENCY,AND OTHER STATISTICS OVER GIVEN CLASS     TAB2
C               INTERVALS.                                               TAB2
C (I) A      IS THE MATRIX OF OBSERVATIONS,NOB ROWS,ONE COLUMN PER VAR.  TAB2
C (I) NOB    IS THE NUMBER OF OBSERVATIONS OF THE VARIABLES.             TAB2
C (I) MRA    IS THE MAXIMUM NUMBER OF ROWS OF A AS DECLARED.             TAB2
C (I) MRB    IS THE MAXIMUM NUMBER OF ROWS OF FREQ AND PCT AS DECLARED.  TAB2
C (I) S      IS A VECTOR OF LENGTH NOB SPECIFYING A SUBSET OF A. ONLY    TAB2
C               OBSERVATIONS CORRESPONDING TO NONZERO S(J) ARE CONSIDERED.TAB2
C (I) NOVAR IS THE VECTOR OF VARIABLES TO BE CROSS-TABULATED. NOVAR(1)   TAB2
C               IS VARIABLE 1 AND NOVAR(2) IS VARIABLE 2.                TAB2
C (I) UBO    IS A 3 BY 2 MATRIX CONTAINING THE LOWER LIMIT,NO.OF         TAB2
C               INTERVALS,AND THE UPPER LIMIT OF VARIABLE I TO BE        TAB2
C               TABULATED IN UBO(1,I),UBO(2,I),UBO(3,I) RESP. (I=1,2).   TAB2
C               NO. OF INTERVALS,UBO(2,I),MUST INCLUDE TWO CELLS FOR     TAB2
C               VALUES BELOW AND ABOVE LIMITS. IF LOWER LIM=UPPER LIM,   TAB2
C               THE PROGRAM USES THE MIN. AND MAX. VALUES OF THE SAMPLE. TAB2
C (R) FREQ   IS THE MATRIX OF FREQUENCIES IN THE TWO-WAY CLASSIFICATION. TAB2
C               DIMENSION OF FREQ IS UBO(2,1) BY UBO(2,2) INTEGER VALUED. TAB2
C (R) PCT    IS THE MATRIX OF PERCENT FREQUENCIES, SAME DIM. AS FREQ.    TAB2
C (R) STAT1 IS THE MATRIX OF SUMMARY STATISTICS- TOTAL,MEAN,VARIANCE     TAB2
C               FOR EACH CLASS INTERVAL OF VARIABLE 1. CLASS INTERVAL J  TAB2
C               CORR.TO COL J. DIMENSION IS 3 BY UBO(2,1) INTEGER VALUED. TAB2
C (R) STAT2 IS THE MATRIX OF SUMMARY STATISTICS FOR VARIABLE 2.          TAB2
C               DIMENSION IS 3 BY UBO(2,2) INTEGER VALUED.               TAB2

      SUBROUTINE AVINT(X,Y,N,XLO,XUP,ANS)                                AVINT   1
C           APPROXIMATE INTEGRATOR OF EXPERIMENTAL DATA TABULATED AT     AVINT
C             ARBITRARILY SPACED ABSCISSA. VALUES OF ABSCISSA ARE ASSUMED AVINT
C             TO BE IN NON-DECREASING ORDER. THE METHOD USED IS BASED ON AVINT
C             OVERLAPPING PARABOLAS.                                     AVINT
C (I) X   IS THE ARRAY OF ABSCISSA AT WHICH FUNCTIONAL VALUES ARE        AVINT
C             SUPPLIED. THESE ABSCISSA MUST BE IN NON-DECREASING ORDER,  AVINT
C             X(I) MUST BE LESS THAN OR EQUAL X(I+1) FOR I=1,2,...,N.    AVINT
C (I) Y   IS THE ARRAY OF VALUES(ORDINATES) OF THE FUNCTION TO BE        AVINT
C             INTEGRATED,I.E., Y(I)=F(X(I)) FOR I=1,2,...,N.             AVINT
C (I) N   IS THE NUMBER OF PAIRS OF VALUES (X(I),Y(I)) PROVIDED. AT      AVINT
C             LEAST THREE X(I) MUST BE BETWEEN XLO AND XUP.              AVINT
C (I) XLO IS THE LOWER LIMIT OF INTEGRATION. XLO MUST BE LESS THAN XUP.  AVINT
C (I) XUP IS THE UPPER LIMIT OF INTEGRATION.                             AVINT
C (R) ANS IS THE VALUE OF THE INTEGRAL AS APPROXIMATED.                  AVINT

      SUBROUTINE SORTXY(X,Y,N)                                           SORTXY 1
C           SORT THE VECTOR X INTO NON-DECREASING ORDER.                 SORTXY
C(IR) X,Y  VECTORS OF REAL VALUES. THE ELEMENTS OF X ARE USED AS KEYS    SORTXY
C              FOR THE SORT. THE ELEMENTS OF Y ARE MOVED SO THAT THEY    SORTXY
C              CORRESPOND TO THE ORIGIONAL VALUES OF X.                  SORTXY
C (I) N     THE NUMBER OF ELEMENTS TO BE SORTED.                         SORTXY
C              IF THE ELEMENTS OF ANOTHER VECTOR Z ARE TO BE MOVED TO    SORTXY
C              CORRESPOND TO THE SORTED VECTOR X THEN THE VECTOR Z       SORTXY
C              MUST BE PASSED TO THIS SUBROUTINE EITHER AS AN ADDITIONAL SORTXY
C              DUMMY VARIABLE OR THRU COMMON AND THE L-TH AND I-TH       SORTXY
C              ELEMENTS MUST BE INTERCHANGED JUST BEFORE STATEMENT 6,I.E. SORTXY
C              T=Z(L) $    Z(L)=Z(I) $      Z(I)=T                       SORTXY
```

```
      SUBROUTINE LABELA(DELX, DELY, XMIN, XMAX, YMIN, YMAX, XFAC, YFAC) LABELA
C     LABELS THE AXES OF PLOTS PRODUCED WITH THE CALCOMP PLOTTER        LABELA
C       SUBROUTINES.                                                    LABELA
C (I) DELX, DELY ARE THE INCREMENT DISTANCES, IN DATA UNITS, ON THE     LABELA
C       X AND Y AXES.                                                   LABELA
C         IF DELX=0, THE X AXIS WILL NOT BE LABELED.                    LABELA
C         IF DELY=0, THE Y AXIS WILL NOT BE LABELED.                    LABELA
C (I) XMIN, XMAX, YMIN, YMAX ARE THE LIMITS, IN DATA UNITS, OVER        LABELA
C       WHICH LABELING IS TO BE DONE.                                   LABELA
C         (THE ARGUMENTS DELX, DELY, XMIN, XMAX, YMIN, AND YMAX,        LABELA
C         ARE USUALLY THE SAME AS THE CORRESPONDING PLTCCA ARGUMENTS    LABELA
C         USED IN PRODUCING THE PLOT.)                                  LABELA
C (I) XFAC, YFAC ARE THE SCALE FACTORS FOR THE X AXIS LABELS AND        LABELA
C       THE Y AXIS LABELS.  IF XFAC IS NOT ZERO, THE X AXIS LABEL       LABELA
C       AT THE POSITION WITH VALUE V WILL HAVE THE VALUE XFAC*V.        LABELA
C       IF XFAC=0, THE LABEL WILL HAVE THE VALUE 10**V.  SIMILARLY      LABELA
C       FOR YFAC AND THE LABELS ON THE Y AXIS.                          LABELA
C         (USUALLY, XFAC=YFAC=1.0)                                      LABELA
C     PLTCCB AND PLTCCS MUST BE CALLED BEFORE EITHER LABELA OR LABELS   LABELA
C       IS CALLED, AND PLACEMENT OF THE LABELS ON THE PAGE WILL BE      LABELA
C       IN REFERENCE TO THE MOST RECENT PLTCCS ENTRY.                   LABELA

      ENTRY LABELS (SIZE, SPACE)                                        LABELS
C     CHANGES THE CONSTANTS 'SIZE' AND 'SPACE' IN THE SUBROUTINE.       LABELS
C       (ONLY IN DECKS DATED            , OR LATER.)                    LABELS
C (I) SIZE IS THE HEIGHT OF THE LABELS, IN INCHES OR CM.                LABELS
C (I) SPACE IS THE DISTANCE, IN INCHES OR CM., FROM YMIN TO THE TOP     LABELS
C       OF THE X AXIS LABELS AND FROM XMIN TO THE RIGHT SIDE OF         LABELS
C       THE Y AXIS LABELS.                                              LABELS
C         IF THIS ENTRY IS NOT CALLED, THE FOLLOWING WILL BE USED....   LABELS
C           SIZE=0.1 INCHES (0.254 CM.), AND                            LABELS
C           SPACE=0.1875 INCHES (0.47625 CM.)                           LABELS

      SUBROUTINE VECTOR (X1, Y1, X2, Y2, I)                             VECTOR
C     PLOTS A VECTOR, WITH ITS HEAD AT (X2, Y2) AND ITS TAIL            VECTOR
C       AT (X1, Y1), USIN THE CALCOMP PLOTTING SUBROUTINE.              VECTOR
C (I) X1, Y1 ARE THE X AND Y COORDINATES OF THE TAIL, IN DATA UNITS.    VECTOR
C (I) X2, Y2 ARE THE X AND Y COORDINATES OF THE HEAD, IN DATA UNITS.    VECTOR
C (I) I=1 MEANS THAT TAIL MARKERS ARE PLOTTED.                          VECTOR
C     I=0 MEANS THAT TAIL MARKERS ARE NOT PLOTTED.                      VECTOR
C     PLTCCB AND PLTCCS MUST BOTH BE CALLED BEFORE EITHER VECTOR        VECTOR
C       OR VECTOS IS CALLED, AND THE SIZE AND POSITION OF THE           VECTOR
C       VECTOR WILL BE IN REFERENCE TO THE MOST RECENT PLTCCS ENTRY.    VECTOR

      ENTRY VECTOS (SIZE, ANGLE)                                        VECTOS
C     CHANGES THE CONSTANTS 'SIZE' AND 'ANGLE' IN THE SUBROUTINE.       VECTOS
C (I) SIZE IS A CONSTANT USED TO DETERMINE THE LENGTH OF THE            VECTOS
C       HEAD AND TAIL MARKERS.                                          VECTOS
C (I) ANGLE IS THE ANGLE, IN RADIANS, BETWEEN THE HEAD AND              VECTOS
C       TAIL MARKERS AND THE BODY OF THE VECTOR.                        VECTOS
C         IF THIS ENTRY IS NOT CALLED, THE FOLLOWING WILL BE USED...    VECTOS
C           SIZE=0.05, AND ANGLE=0.5235988 (30.0 DEGREES).              VECTOS
```

```
      SUBROUTINE DASHL (PA, N, X, Y, M)                                 DASHL
C     PLOT DASHED LINES, AS DEFINED BY THE USER, ALONG A CURVE,         DASHL
C       USING THE CALCOMP PLOTTING SUBROUTINE.                          DASHL
C (I) PA IS THE NAME OF THE LINEAR ARRAY WHICH CONTAINS THE LENGTH      DASHL
C       OF THE DASHES AND SPACES, IN INCHES OR CM., MAKING UP THE       DASHL
C       PATTERN WHICH WILL BE REPEATED UNTIL THE CURVE IS DRAWN.        DASHL
C         PA(I), FOR I=1,3,5,....., CORRESPOND TO THE DASHES IN THE     DASHL
C           PATTERN, WHILE                                              DASHL
C         PA(I), FOR I=2,4,6,....., CORRESPOND TO THE SPACES            DASHL
C           BETWEEN THE DASHES.                                         DASHL
C (I) N IS THE NUMBER OF NUMBERS IN THE PATTERN DESCRIPTION.            DASHL
C (I) X, Y ARE THE NAMES OF LINEAR ARRAYS WHICH CONTAIN THE             DASHL
C       X AND Y COORDINATES OF THE POINTS DESCRIBING THE CURVE.         DASHL
C (I) M IS THE NUMBER OF POINTS IN THE CURVE DESCRIPTION.               DASHL
C     PLTCCB AND PLTCCS MUST BE CALLED BEFORE EITHER DASHL OR           DASHL
C       DASHLA IS CALLED, AND PLOTTING IS DONE WITH REFERENCE           DASHL
C       TO THE MOST RECENT PLTCCS ENTRY.                                DASHL

      ENTRY DASHLA (PA, N, X, Y, M, XMIN, XMAX, YMIN, YMAX)             DASHLA
C (I) XMIN, XMAX, YMIN, YMAX ARE THE GRAPH LIMITS, IN DATA UNITS.       DASHLA
C       THE DASHLA ENTRY WILL NOT PLOT OUTSIDE THESE LIMITS.            DASHLA
C         (THE DASHL ENTRY USES THE BOARD LIMITS.)                      DASHLA

      SUBROUTINE EIGEN(A,N,NM,T,EVR,EVI,VECR,VECI,INDIC)                EIGEN  1
C           FINDS ALL THE EIGENVALUES AND EIGENVECTORS OF A REAL        EIGEN
C             GENERAL MATRIX OF ORDER N. THE REAL EIGENVECTOR IS        EIGEN
C             NORMALIZED SO THAT THE SUM OF THE SQUARES OF THE          EIGEN
C             COMPONENTS IS EQUAL TO ONE. THE COMPLEX EIGENVECTOR IS    EIGEN
C             NORMALIZED SO THAT THE COMPONENT WITH THE LARGEST VALUE   EIGEN
C             IN MODULUS HAS ITS REAL PART EQUAL TO ONE AND THE         EIGEN
C             IMAGINARY PART EQUAL TO ZERO. THE ORIGINAL MATRIX A IS    EIGEN
C             DESTROYED BY THE SUBROUTINE.                              EIGEN
C (I) A     IS THE MATRIX OF ORDER N.                                   EIGEN
C (I) N     IS THE ORDER OF THE MATRIX.                                 EIGEN
C (I) NM    IS THE FIRST DIMENSION OF THE MATRICES A,VECR,VECI AND THE  EIGEN
C             DIMENSION OF THE VECTORS EVR,EVI AND INDIC AS DECLARED.   EIGEN
C (I) T     IS A PARAMETER EQUAL TO THE NO. OF BINARY DIGITS IN THE     EIGEN
C             MANTISSA OF A FLOATING POINT NO., FOR BRLESC, T=53.0      EIGEN
C (R) EVR   IS A VECTOR CONTAINING THE REAL PARTS OF THE N COMPUTED     EIGEN
C             EIGENVALUES.                                              EIGEN
C (R) EVI   IS A VECTOR CONTAINING THE IMAGINARY PARTS OF THE N         EIGEN
C             COMPUTED EIGENVALUES.                                     EIGEN
C (R) VECR  IS A MATRIX,WHERE COLUMN I CONTAINS THE REAL COMPONENTS OF  EIGEN
C             THE NORMALIZED EIGENVECTOR I CORRESPONDING TO THE         EIGEN
C             EIGENVALUE STORED IN EVR(I) AND EVI(I) (I=1,2,...,N).     EIGEN
C (R) VECI  IS A MATRIX,WHERE COLUMN I CONTAINS THE IMAGINARY COMPONENTSEIGEN
C             OF THE NORMALIZED EIGENVECTOR I CORRESPONDING TO THE      EIGEN
C             EIGENVALUE STORED IN EVR(I) AND EVI(I) (I=1,2,...,N).     EIGEN
C (R) INDIC IS A VECTOR INDICATING THE SUCCESS OF THE SUBROUTINE EIGEN  EIGEN
C             AS FOLLOWS                                                EIGEN
C               VALUE OF INDIC(I)    EIGENVALUE I    EIGENVECTOR I      EIGEN
C                      0               NOT FOUND       NOT FOUND        EIGEN
C                      1               FOUND           NOT FOUND        EIGEN
C                      2               FOUND           FOUND            EIGEN
```

```
      SUBROUTINE BES(X,NO,KODE,RSLT1,RSLT2,T1,T2)                        BES    1
C           CALCULATES THE BESSEL FUNCTIONS J(X),Y(X),I(X), OR K(X) FOR BES
C             REAL ARGUMENTS AND INTEGER ORDERS.                         BES
C (I) X     IS THE REAL ARGUMENT OF THE BESSEL FUNCTION. THE ARGUMENT   BES
C             MAY BE POSITIVE,ZERO, OR NEGATIVE(NEG. ARG. FOR Y(X) OR   BES
C             K(X) PRODUCES ERROR MESSAGE SINCE RESULTS MAY BE COMPLEX) BES
C             RESTRICTION ON RANGE IS                                    BES
C               FOR J(X), -660.0 .LE. X .LE. 660.0                       BES
C               FOR Y(X), 0.0 .LE. X .LE. 660.0                          BES
C               FOR I(X), -350.0 .LE. X .LE. 350.0                       BES
C               FOR K(X), 0.0 .LE. X .LE. 350.0                          BES
C (I) NO    IS THE INTEGER ORDER OF FUNCTION DESIRED FOR A SINGLE VALUE BES
C             TO BE RETURNED, OR THE MAXIMUM ORDER DESIRED (+ OR -) IF  BES
C             AN ARRAY OF VALUES IS TO BE RETURNED.                      BES
C             LET XX = ABS(X). THEN BOUNDS ON ORDERS ARE                 BES
C             1. FOR 0.0 .LE. XX .LE. 50.0                               BES
C                  ABS(NO) .LE. INT(2.50*XX + 90.0)                      BES
C             2. FOR 50.0 .LT. XX .LE. 150.0                             BES
C                  ABS(NO) .LE. INT(1.67*XX + 150.0)                     BES
C             3. FOR 150.0 .LT. XX .LE. 300.0                            BES
C                  ABS(NO) .LE. INT(1.43*XX + 200.0)                     BES
C             4. FOR 300.0 .LT. XX .LE. 660.0                            BES
C                  ABS(NO) .LE. INT(1.25*XX + 240.0)                     BES
C (I) KODE  IS THE INTEGER INDICATOR FOR THE PARTICULAR FUNCTION TO BE  BES
C             COMPUTED.                                                  BES
C               KODE = 10 --FUNCTION J(X) ONLY                           BES
C                    = 11 --          Y(X) ONLY                          BES
C                    = 12 --          J(X) AND Y(X)                      BES
C                    = 20 --          I(X) ONLY                          BES
C                    = 21 --          K(X) ONLY                          BES
C                    = 22 --          I(X) AND K(X)                      BES
C (R) RSLT1 IS THE REAL FUNCTION VALUE FOR J(X) OR I(X) CORRESPONDING   BES
C             TO THE ORDER AND ARGUMENT SUPPLIED, DEPENDING ON THE KODE BES
C             VALUE. THIS PARAMETER WOULD CONTAIN THE RESULT IF ONLY    BES
C             ONE FUNCTION VALUE IS TO BE RETURNED.                      BES
C (R) RSLT2 CONTAINS THE REAL FUNCTION VALUE FOR Y(X) OR K(X) IN A      BES
C             MANNER SIMILAR TO RSLT1.                                   BES
C (R) T1    IS A WORK AREA WHICH WILL CONTAIN THE ARRAY OF REAL FUNCT-  BES
C             ION VALUES FOR J(X) OR I(X) OF ORDERS ZERO THROUGH NO,    BES
C             DEPENDING ON KODE. T1 MUST BE DIMENSIONED IN THE CALLING  BES
C             PROGRAM AND MUST CONTAIN AT LEAST M CELLS OF STORAGE,     BES
C             WHERE                                                      BES
C                    M = MAX(ABS(NO),INT(2*ABS(X))) + 51                 BES
C             IN USING THE ARRAY, T1(1) = FUNCTION OF ORDER 0,...,      BES
C             T1(NO+1) = FUNCTION OF ORDER NO.                           BES
C (R) T2    IS SIMILAR TO T1 FOR THE FUNCTIONS Y(X) OR K(X). AN EXCEPT- BES
C             ION IS THAT IF ONLY J(X) OR I(X) ARE CALLED, THEN T2      BES
C             NEEDS NO DIMENSION IN THE CALLING PROGRAM, BUT THE PARAM- BES
C             ETER MUST STILL APPEAR IN THE CALLING SEQUENCE. OTHERWISE BES
C             T2 MUST BE DIMENSIONED AT LEAST M.                         BES
```

Unclassified
Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) U.S. Army Aberdeen Research and Development Center Aberdeen Proving Ground, Maryland	2a. REPORT SECURITY CLASSIFICATION Unclassified
	2b. GROUP

3. REPORT TITLE

BRLESC I/II FORTRAN

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

5. AUTHOR(S) (Last name, first name, initial)

Campbell, Lloyd W. and Beck, Glenn A.

6. REPORT DATE March 1970	7a. TOTAL NO. OF PAGES 163	7b. NO. OF REFS 6

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S) ARDC Technical Report No. 5
b. PROJECT NO. Funded by all ARDC RDT&E Projects	
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	

10. AVAILABILITY/LIMITATION NOTICES

This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY U.S. Army Materiel Command Washington, D.C.

13. ABSTRACT

FORTRAN is a popular scientific programming language that has been implemented on many computers. This report describes the FORTRAN language in general and includes specific details about its implementation on the BRLESC I and BRLESC II computers at the Aberdeen Research and Development Center.

DD FORM 1 JAN 64 1473

Unclassified
Security Classification

Unclassified
Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
FORTRAN Programming Language Digital Computer BRLESC I Computer BRLESC II Computer Compiler						

INSTRUCTIONS

1. ORIGINATING ACTIVITY: Enter the name and address of the contractor, subcontractor, grantee, Department of Defense activity or other organization *(corporate author)* issuing the report.

2*a*. REPORT SECURITY CLASSIFICATION: Enter the overall security classification of the report. Indicate whether "Restricted Data" is included. Marking is to be in accordance with appropriate security regulations.

2*b*. GROUP: Automatic downgrading is specified in DoD Directive 5200.10 and Armed Forces Industrial Manual. Enter the group number. Also, when applicable, show that optional markings have been used for Group 3 and Group 4 as authorized.

3. REPORT TITLE: Enter the complete report title in all capital letters. Titles in all cases should be unclassified. If a meaningful title cannot be selected without classification, show title classification in all capitals in parenthesis immediately following the title.

4. DESCRIPTIVE NOTES: If appropriate, enter the type of report, e.g., interim, progress, summary, annual, or final. Give the inclusive dates when a specific reporting period is covered.

5. AUTHOR(S): Enter the name(s) of author(s) as shown on or in the report. Enter last name, first name, middle initial. If military, show rank and branch of service. The name of the principal author is an absolute minimum requirement.

6. REPORT DATE: Enter the date of the report as day, month, year; or month, year. If more than one date appears on the report, use date of publication.

7*a*. TOTAL NUMBER OF PAGES: The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.

7*b*. NUMBER OF REFERENCES: Enter the total number of references cited in the report.

8*a*. CONTRACT OR GRANT NUMBER: If appropriate, enter the applicable number of the contract or grant under which the report was written.

8*b*, 8*c*, & 8*d*. PROJECT NUMBER: Enter the appropriate military department identification, such as project number, subproject number, system numbers, task number, etc.

9*a*. ORIGINATOR'S REPORT NUMBER(S): Enter the official report number by which the document will be identified and controlled by the originating activity. This number must be unique to this report.

9*b*. OTHER REPORT NUMBER(S): If the report has been assigned any other report numbers *(either by the originator or by the sponsor)*, also enter this number(s).

10. AVAILABILITY/LIMITATION NOTICES: Enter any limitations on further dissemination of the report, other than those imposed by security classification, using standard statements such as:

(1) "Qualified requesters may obtain copies of this report from DDC."

(2) "Foreign announcement and dissemination of this report by DDC is not authorized."

(3) "U. S. Government agencies may obtain copies of this report directly from DDC. Other qualified DDC users shall request through
_______________."

(4) "U. S. military agencies may obtain copies of this report directly from DDC. Other qualified users shall request through
_______________."

(5) "All distribution of this report is controlled. Qualified DDC users shall request through
_______________."

If the report has been furnished to the Office of Technical Services, Department of Commerce, for sale to the public, indicate this fact and enter the price, if known.

11. SUPPLEMENTARY NOTES: Use for additional explanatory notes.

12. SPONSORING MILITARY ACTIVITY: Enter the name of the departmental project office or laboratory sponsoring *(paying for)* the research and development. Include address.

13. ABSTRACT: Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may also appear elsewhere in the body of the technical report. If additional space is required, a continuation sheet shall be attached.

It is highly desirable that the abstract of classified reports be unclassified. Each paragraph of the abstract shall end with an indication of the military security classification of the information in the paragraph, represented as *(TS)*, *(S)*, *(C)*, or *(U)*.

There is no limitation on the length of the abstract. However, the suggested length is from 150 to 225 words.

14. KEY WORDS: Key words are technically meaningful terms or short phrases that characterize a report and may be used as index entries for cataloging the report. Key words must be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context. The assignment of links, rules, and weights is optional.

Unclassified
Security Classification